M A S T E R A R B E I T

Constructing OLAP hierarchies from Statistical Linked Data

von
Dominik Siegele

eingereicht am 17.02.2012 beim
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
des Karlsruher Instituts für Technologie

Referent: Prof. Dr. Rudi Studer
Betreuer: Dipl.-Inform. Benedikt Kämpgen

Eidesstattliche Versicherung

Ich, Dominik Siegele, versichere hiermit eidesstattlich, dass ich die hier vorliegende Arbeit mit dem Titel

Constructing OLAP hierarchies from Statistical Linked Data

selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder inhaltlich übernommene Stellen habe ich als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung habe ich beachtet.

Karlsruhe, den 17.02.2012 _____

Vorwort

Mit diesem Vorwort möchte ich allen Personen danken, die mich bei der Erstellung dieser Masterarbeit betreut und unterstützt haben:

Am Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) der Fakultät für Wirtschaftswissenschaften des Karlsruher Instituts für Technologie gilt mein Dank den Mitarbeitern der Forschungsgruppe Wissensmanagement, die mir die Bearbeitung dieses Themas ermöglichten und mit konstruktiver Kritik zum Gelingen dieser Arbeit beigetragen haben.
Besonderer Dank gilt hierbei dem Forschungsgruppenleiter *Prof. Dr. Rudi Studer* sowie meinem Betreuer *Benedikt Kämpgen*, der mir jederzeit mit Rat und Tat zur Seite stand.

Weiterhin möchte ich mich bei meiner Familie und meiner Freundin *Caroline* für die Unterstützung während meines gesamten Studiums danken.

Zuletzt gilt mein Dank meinen Kommilitonen sowie allen sonstigen Personen, die diese Arbeit und mein Studium auf irgendeine Art und Weise unterstützt haben.

Contents

1 Introduction 10
1.1 Problem description 10
1.2 Aims and organization of this thesis 12
1.3 Motivation 14
2 Theoretical background 16
2.1 Online Analytical Processing 16
2.1.1 Conceptual Model of OLAP 17
2.1.1.1 Data Cube 17
2.1.1.2 Aggregation functions and summarizability 18
2.1.2 Business Intelligence 21
2.1.2.1 BI process 21
2.1.2.2 Facets of BI 22
2.1.2.3 Data – Information - Knowledge 23
2.1.3 Data Warehouse 24
2.1.3.1 ETL process 25
2.1.3.2 Star and snowflake schema 26
2.2 Linked Data 28
2.2.1 RDF, RDFS and OWL 29
2.2.2 RDF Data Cube Vocabulary and SDMX 31
2.2.3 SPARQL 33
2.2.4 Semantic Web 34
2.2.5 Linking Open Data Cloud and Statistical Linked Data sets 35
2.2.6 Linked Data as a source for data warehouses 38
2.3 Hierarchies 39
2.3.1 Individual hierarchies 42
2.3.1.1 Simple hierarchies 42
2.3.1.1.1 Type of simple hierarchies 42
2.3.1.1.1.1 Symmetric hierarchies 42
2.3.1.1.1.2 Asymmetric hierarchies 43
2.3.1.1.1.3 Generalized hierarchies 43

2.3.1.1.2 Strictness of simple hierarchies 45
2.3.1.1.2.1 Strict hierarchies 45
2.3.1.1.2.2 Non-strict hierarchies 45
2.3.1.2 Multiple alternative hierarchies 46
2.3.2 Parallel hierarchies 47
2.3.2.1 Parallel independent hierarchies 47
2.3.2.2 Parallel dependent hierarchies 47
2.3.3 Usefulness of hierarchies 48
2.3.3.1 Metrics corresponding to one approach 49
2.3.3.2 Metrics corresponding to one found hierarchy 50
3 Expressing OLAP hierarchies in RDF 53
3.1 Simple Knowledge Organization System (SKOS) 53
3.2 Weaknesses of SKOS 54
3.3 Proposed extensions to SKOS 57
3.4 Resulting vocabulary 60
4 Transforming Linked Data into OLAP hierarchies 63
4.1 OLAP4J 63
4.2 Mapping between SKOS and OLAP4J 64
4.2.1 Transforming parallel hierarchies in individual hierarchies 66
4.2.2 Transforming multiple alternative hierarchies in simple hierarchies 67
4.2.3 Transforming generalized hierarchies 67
4.3 OLAP4J methods 68
4.3.1 getHierarchies 70
4.3.2 getLevels 74
4.3.3 getMembers 75
5 Approaches for learning OLAP hierarchies from RDF 78
5.1 Steps of the approaches 78
5.1.1 Start 79
5.1.2 Creating a concept scheme 79
5.1.3 Loading relevant triples 80
5.1.4 Adding hierarchical information 80
5.1.5 End of the approach reached? 81
5.1.6 End 81
5.2 Technical implementation 82

5.3 Specific approaches 82
5.3.1 Approach ‘time’ 82
5.3.1.1 Overview 82
5.3.1.2 Algorithm 85
5.3.1.2.1 Start 85
5.3.1.2.2 Creating a concept scheme 86
5.3.1.2.3 Loading relevant triples 91
5.3.1.2.4 Adding hierarchical information 91
5.3.1.2.5 End of the approach reached? 96
5.3.1.2.6 End 96
5.3.1.3 Example 96
5.3.1.4 Resulting hierarchies 97
5.3.1.5 Criticism 97
5.3.2 Approach ‘geo’ 99
5.3.2.1 Overview 99
5.3.2.2 Algorithm 100
5.3.2.2.1 Start 100
5.3.2.2.2 Creating a concept scheme 100
5.3.2.2.3 Loading relevant triples 102
5.3.2.2.4 Adding hierarchical information 102
5.3.2.2.5 End of the approach reached? 105
5.3.2.2.6 End 105
5.3.2.3 Example 106
5.3.2.4 Resulting hierarchies 106
5.3.2.5 Criticism 106
5.4 Generic approaches 108
5.4.1 Approach ‘rdfs:subClassOf’ 108
5.4.1.1 Overview 108
5.4.1.2 Parameters 108
5.4.1.3 Algorithm 110
5.4.1.3.1 Start 110
5.4.1.3.2 Creating a concept scheme 110
5.4.1.3.3 Loading relevant triples 111
5.4.1.3.4 Adding hierarchical information 112

5.4.1.3.5 End of the approach reached? ... 114
5.4.1.3.6 End ... 114
5.4.1.4 Example ... 114
5.4.1.5 Resulting hierarchies ... 117
5.4.1.6 Criticism ... 118
5.4.2 Approach 'properties' ... 119
5.4.2.1 Overview ... 119
5.4.2.2 Parameters ... 119
5.4.2.3 Algorithm ... 121
5.4.2.3.1 Start ... 121
5.4.2.3.2 Creating a concept scheme ... 121
5.4.2.3.3 Loading relevant triples ... 122
5.4.2.3.4 Adding hierarchical information ... 123
5.4.2.3.5 End of the approach reached? ... 127
5.4.2.3.6 End ... 127
5.4.2.4 Example ... 127
5.4.2.5 Resulting hierarchies ... 130
5.4.2.6 Criticism ... 131
6 Evaluation ... 132
6.1 Approach 'time' ... 132
6.2 Approach 'geo' ... 133
6.3 Approach 'rdfs:subClassOf' ... 134
6.4 Approach 'properties' ... 136
7 Conclusions ... 140
7.1 Summary ... 140
7.2 Resulting lessons learned ... 141
7.3 Future research topics ... 144

List of figures

Figure 1: Multidimensional data model: cube [modified from KeMU06, p.95] 17
Figure 2: A generic Business Intelligence process [modified from Humm08] 22
Figure 3: Facets of Business Intelligence [KeMU06, p.4] 23
Figure 4: Stairs of knowledge [modified from Nort05, p.32] 23
Figure 5: ETL-process 25
Figure 6: Star schema 27
Figure 7: Snowflake schema 28
Figure 8: Triple consisting of subject, predicate and object 29
Figure 9: RDF Data Cube vocabulary [CyRT10] 32
Figure 10: Semantic Web layer cake 35
Figure 11: Linking Open Data cloud diagram, September 2011 [CyJe11] 36
Figure 12: Metamodel of hierarchy classification 41
Figure 13: Notation of hierarchies in a multidimensional model: (a) level, (b) hierarchy, (c) cardinalities, (d) analysis criterion, and (e) fact relationship 41
Figure 14: Example for a symmetric hierarchy: (a) model and (b) instances 42
Figure 15: Example for an asymmetric hierarchy: (a) schema and (b) instances 43
Figure 16: Example for a generalized hierarchy: (a) schema and (b) instances 44
Figure 17: Example for a non-covering hierarchy 44
Figure 18: Example for a symmetric non-strict hierarchy: (a) model and (b) instances 45
Figure 19: Example for a multiple alternative hierarchy: (a) model and (b) relations 46
Figure 20: Example for a parallel independent hierarchy 47
Figure 21: Example for a parallel dependent hierarchy: (a) model and (b) relations 48
Figure 22: Example of a hierarchy to explain the formulas 51
Figure 23: Hierarchies expressible with SKOS 56
Figure 24: Example for a transformation of a parallel independent hierarchy 66
Figure 25: Example for a transformation of a parallel dependent hierarchy 67
Figure 26: Example for a transformation of a multiple alternative hierarchy 67
Figure 27: Example for a transformation of a generalized hierarchy 68
Figure 28: Steps of the approaches 79
Figure 29: Examples for specific hierarchies of the time dimension: schema 83
Figure 30: Resulting multiple alternative hierarchy of the approach ‘Time’: schema 84
Figure 31: Simplified example for the approach 'rdfs:subClassOf': triples 115
Figure 32: Resulting hierarchy of the example with setting the flag: schema 117
Figure 33: Resulting hierarchy of the example without setting the flag: schema 117
Figure 34: Simplified example for the approach 'properties': triples 128
Figure 35: Resulting hierarchy of the approach 'time': schema 132
Figure 36: Resulting hierarchy of the approach 'geo': schema 133
Figure 37: Resulting hierarchy of the approach 'subClassOf': schema 135
Figure 38: Resulting hierarchy of the approach 'properties': schema 138

List of tables

Table 1: Consistency [modified from LeSh97] 21
Table 2: Examples of data sets using the RDF Data Cube Vocabulary 37
Table 3: Formal notation for a hierarchy 40
Table 4: Metrics of a simple hierarchy depending on type and strictness 50
Table 5: Classes and properties to express all proper OLAP hierarchies in RDF 61
Table 6: Hierarchy mapping between SKOS and OLAP4J 65
Table 7: Temporal data types 87
Table 8: Levels of the NUTS classification system 99
Table 9: Example for the approach 'geo' 106
Table 10: Mapping between ontology and hierarchy of the approach 'rdfs:subClassOf' 109
Table 11: Mapping between triple and hierarchy of the approach ‘properties’ 119
Table 12: Components for the evaluation of the approach 'time' 132
Table 13: Components for the evaluation of the approach 'geo' 133
Table 14: Components for the evaluation of the approach 'rdfs:subClassOf' 134
Table 15: Levels of the approach 'rdfs:subClassOf' 135
Table 16: Components for the evaluation of the approach 'properties' 136
Table 17: Candidates for levels of the approach 'properties' 137
Table 18: Resulting hierarchies of each approach 141

1 Introduction

First of all, the following subsections introduce in this master thesis by describing the problem situation, explaining the aims of this work and giving a motivation.

1.1 Problem description

The choice behavior in all areas of life is essentially determined by the available information. To make intelligent and reasonable decisions information is required, which can be obtained from relevant data.

On the one hand there is the trend that the available data is growing always faster and faster. For example the research company Gartner, Inc. reports that data growth is the biggest challenge in 2011 for large-enterprise data centers [Gart10]. Another up to date example for publishing data is the German census of population in 2011, where about 10% of the population is interviewed to acquire data for important administrative and scientific objectives [Wagn10]. On the other hand, decision makers like managers in companies or representatives in politics threat to be overflowed with too much information. However, they complain about less information, because they have the feeling that they do not have the relevant information to make the right decision [GlGD08, S. 32].

Considering both aspects, there is an information deficiency simultaneous to an increasing data appearance, e.g. in companies [GaGP09, S. 41]. This means that there is much data available but in fact only less data can be used. The right decision-making basis would be the relevant information, no more, no less. To retrieve the relevant information out of the mass of data, there are some requirements for content and structure of this data, so that the relevant information can be extracted in an effective and efficient way.

Online analytical processing (OLAP) has become a popular technique to support the process from retrieving information out of data [ChDa97]. The advantage of OLAP is that observations (so called measures) can be viewed from different perspectives (so called dimensions) and on different levels of detail (so called levels of hierarchies). A

possible format for OLAP is OLAP4J[1], an open Java application programming interface (API) for accessing multi-dimensional data.

Usually, a data warehouse (DWH) is used where relevant data is provided for analytical purposes. To do such analysis this large data repository has to be filled with relevant data. This is done by the extract-transform-load (ETL) process, where data is first extracted from different data sources, second transformed to suffice the OLAP technique and third loaded, which means stored persistently in the DWH [ChDa97].

One possibility to acquire data for a DWH is to extract published data from the web, which exists there in different formats [PBAP08]. Linked Data is an approach, to standardize the format for published data so that it can be interconnected on the web [BiHB09]. The main idea is that each object of the real world (so called resource) has a unique resource identifier (URI) and resources can be linked to other resources for what the markup language Resource Description Framework (RDF) is used. One special format for multi-dimensional data, such as statistics, is the RDF Data Cube Vocabulary, which is compatible to RDF [CyRT10].

Regarding the process in reference to the above described standards, the data can be extracted from the web, transformed from the RDF Data Cube vocabulary to the OLAP4J format and loaded into a DWH. In so doing, the main problem is the transformation of the data, which should work automatically without any manually effort. One approach of an automatic transformation is based on a mapping between the RDF Data Cube Vocabulary and a Multidimensional Data Model [KäHa11].

Although this approach provides the possibility to automatically transform data sets to the Extensible Markup Language for Analysis (XMLA)[2] format, which is an alternative OLAP format to OLAP4J, some questions are still unanswered. One aspect that is not solved at all is to deal with hierarchies, because the dimensions of the transformed data in this approach have a flattened structure without any hierarchical relationships. Since the data in the RDF Data Cube Vocabulary already has a certain semantic, the authors of this approach of an automatic transformation believe that there are more possibilities to find meaningful hierarchies in Statistical Linked Data

[1] http://olap4j.svn.sourceforge.net/viewvc/olap4j/trunk/doc/olap4j_fs.html
[2] http://news.xmlforanalysis.com/what-is-xmla

[KäHa11]. For example hierarchical structures of the relationship between resources could be used for the automatic formation of hierarchies. As a consequence, the question how hierarchies that can be used in OLAP systems can be constructed from Statistical Linked Data is not answered yet.

1.2 Aims and organization of this thesis

The aim of this master thesis is to support the extract-transform-load (ETL) process from the RDF Data Cube Vocabulary to the OLAP data model, so that it can proceed as much automated as possible. The semantics of the data should be automatically interpreted and an OLAP data cube should be constructed. The main focus here is on hierarchies. This means that this master thesis researches, how useful hierarchies can be constructed automatically. To build on existing standards, it is assumed that the RDF data is in the format of the RDF Data Cube Vocabulary and has to be transformed in the specific OLAP4J standard of the multidimensional data model.

The concept of this thesis is successive and the sections are based on each other. The organization of this thesis also corresponds to work procedure, thus the chronological and logical sequence of the steps. To construct hierarchies out of Statistical Linked data and use them in OLAP systems, the following questions have to be answered. They are the essential parts of this master thesis. Each question is answered in an own section after this section 1 ‘Introduction’, which also serves as motivation for this thesis:

- Which are useful Statistical Linked Data hierarchies?
  First of all, to understand the relevant concepts, technologies and standards and to know the possible hierarchies a qualified background is required. For this reason this master thesis introduces the relevant concepts, technologies and standards theoretically. To measure the usefulness of a hierarchy, a catalog of relevant metrics is developed. Hence, this question is answered in section 2 ‘Theoretical background’.
- How can Statistical Linked Data hierarchies be expressed?
  Linked Data publishers should have the possibility to explicitly express all possible OLAP hierarchies with a firm and standardized Vocabulary in Linked

Data. Since there is already a recommended way of expressing hierarchies with the RDF Data Cube Vocabulary, this master thesis critically validates this vocabulary. Furthermore it proposes an extension for this vocabulary to have the possibility to express all different types of hierarchies. This vocabulary is introduced in section 3 ‘Expressing OLAP hierarchies in RDF’.

- How can Statistical Linked Data hierarchies be used?
  There is a mapping required which transforms the explicit hierarchies, expressed in a firm and standardized vocabulary in Linked Data into an OLAP application to analyze the data. This master thesis describes the developed algorithms to support the OLAP4J standard. This developed driver queries against Statistical Linked Data, including explicitly expressed hierarchies and transforms them into the standardized format for common OLAP applications. This transformation of Linked Data into OLAP hierarchies is done in section 4 ‘Transforming Linked Data into OLAP hierarchies’.
- How can Statistical Linked Data hierarchies be constructed?
  If published data sets do not include explicit hierarchical information, there are algorithms required to make implicit included hierarchies explicit. There may be generic algorithms that potentially work for all data and also specific algorithms which are customized for specific data. This master thesis proposes several developed approaches to construct hierarchies from Statistical Linked Data. There are specific approaches (time and geo) for often occurring data and also generic approaches (rdfs:subClassOf and properties), which can potentially be applied to all data. Considering these approaches, this master thesis has to cope with the following challenges:
    - How can hierarchies be constructed and what can be names for them?
    - How can levels be constructed and what can be names for them?
    - How can members be constructed and what can be names for them?

  These approaches are explained in section 5 ‘Approaches for learning OLAP hierarchies from RDF’.

On this basis, section 6 ‘Evaluation’ tests the functionality of the different approaches on real data. The last part of this thesis is section 7 ‘Conclusions’, where the resulting lessons learned are described and a perspective for future work is given.

1.3 Motivation

Besides the general advantages of the transformation of Statistical Linked Data to OLAP cubes, the generation of hierarchies during this process, which is the main focus in this master thesis, enables the following special advantages respectively use cases:

- Automatic construction of hierarchies:
  Linked Data is used to automatically construct hierarchies, which can then be used in OLAP systems. Therefore several approaches are developed. This means that no manually effort is necessary to generate hierarchies, because the approaches can be used.
- Making hierarchies explicit:
  Because of the application of algorithms on existing Linked Data, implicit included hierarchical structures become explicit. This means that the hierarchies are expressed in a standardized way so that the hierarchical structures are clear.
- Additional hierarchies:
  With the generation of additional hierarchies with new levels, new members and new allocation of members to particular levels, the use of hierarchies becomes more useful, because the data can be analyzed on more levels of detail so that more and new knowledge can be derived out of this data.
- Additional types of hierarchies:
  Besides the possibility to generate additional hierarchies with new levels and allocation of members to particular levels, there also may arise new types of hierarchies for a particular dimension. With these new types of hierarchies, the data can be analyzed in another way, so that more and new knowledge can be derived out of this data.
- Integration of more and less aggregated data:
  Assuming that there are two or more data sets, which share at least one common dimension but on different levels of detail, the transformation into a cube and generation of hierarchies enables the possibility to analyze this data together by using a common hierarchy level. The following cases can be distinguished:

- Some given dimension members of different data sets roll up to the same member in a higher level of detail. For example within the time dimension the member ‘December 2011’ is used in one data set and the member ‘August 2011’ is used in another data set. Both data sets can be integrated by using a hierarchy, which includes the two given members and the additional member ‘Year 2011’. Both given members roll up to the new member ‘Year 2011’.
- Some given dimension members of one data set roll up to given members of another data set. For example within the time dimension the member ‘August 2011’ is used in one data set and the member ‘Year 2011’ is used in another data set. Both data sets can be integrated by using a hierarchy, which includes these two given members and the member ‘August 2011’ rolls up to the ‘Year 2011’.

2 Theoretical background

This thesis is called ‘Constructing OLAP hierarchies from Statistical Linked Data‘. If you look closely to this title, you will notice that mainly the following topics are relevant:

- OLAP
- Hierarchies
- Statistical Linked Data

Therefore these concepts and technologies and their relevant surround are explained in the following subsections, to give the reader of this thesis a qualified theoretical background.

2.1 Online Analytical Processing

The term Online Analytical Processing (OLAP) was established by Codd et al., who define OLAP as ‘the dynamic enterprise analysis required to create, manipulate, animate, and synthesize information from exegetical, contemplative, and formulaic data analysis models. This includes the ability to discern new or unanticipated relationships between variables, the ability to identify the parameters necessary to handle large amounts of data, to create an unlimited number of dimensions (consolidation paths), and to specify cross-dimensional conditions and expressions’ [CoCS93].

Considering their idea, the term OLAP is representing two facts. On the one hand it emphasizes the analytical processing, where information is gained out of data, in contrast to the transaction processing, where operational data is generated, e.g. entering orders in a company, credit transfer in a bank [ChGl06, p. 145]. On the other hand it emphasizes the interactive and multidimensional analysis on historic and consolidated data [FBSV00, p. 88].

2.1.1 **Conceptual Model of OLAP**

2.1.1.1 Data Cube

On the conceptual level, the data model of OLAP can be represented as a cube.
The essential feature of this cube is to subdivide the data into a set of facts (cells of the cube) and dimensions (edges of the cube). The facts represent key figures, which means the values that are assigned to a tuple. The dimensions relate the key figures to properties or objective criteria. Therefore each dimension is described by a set of attributes and the dimensions together uniquely determine the measures. Since the cube in the following figure contains exactly three dimensions, the number of dimensions in the multidimensional data model is unrestricted, so that a hypercube is generated.

To provide information on different aggregation levels, hierarchies can be defined over the dimensions. The cells are determined by the selected hierarchies and levels of each dimension, so that a measure can be considered along one dimension on different hierarchical levels [ChDa97].

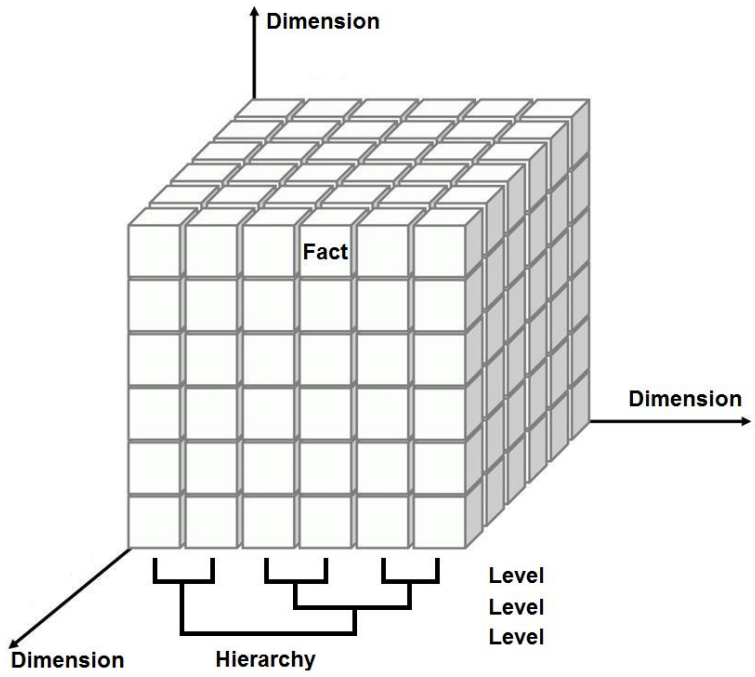

Figure 1: Multidimensional data model: cube [modified from KeMU06, p.95]

There are several popular operations that can be done with an OLAP cube to regard the contained data from the desired perspective [ChDa97]:

- Slicing/Dicing: Reducing the dimensionality of the data by taking a projection of the data on a subset of dimensions for selected data of other dimensions.
- Pivoting/Rotating: Re-orienting the multidimensional view of data.
- Roll-Up/Drill-Down: Navigating along a hierarchy to look at the data on a more general resp. specific level. For these operations, hierarchies are used.
- Ranking: Sorting the data.

These operations can be executed by querying multidimensional data. Therefore, the declarative query language Multidimensional Expressions (MDX) can be used, which is a part of Microsoft's OLAP product[3] [NiNT01].

2.1.1.2 Aggregation functions and summarizability

In this master thesis, the focus is on generating hierarchies. It doesn't make a statement of correct aggregation of the measures, which is the next step, when possible hierarchies are found. This means by implication, the process of finding hierarchies in this master thesis is independent of aggregation functions and summarizability conditions. If correct aggregation aspects were taken into account in this master thesis, the algorithms to generate hierarchies possibly have to be adapted. For example depending on the resulting hierarchy type, additional information concerning correct summarizability has to be provided. For the sake of completeness, the problematic of aggregating data is broached in this section. So it may not be forgotten that not only hierarchies but also aggregation functions play an important role, when data is summarized in a higher level of detail.

As described above, OLAP provides the possibility, to view the data of a cube in different perspectives (dimensions) and on different levels of detail (hierarchies). The advantage of this concept is that data can be viewed not on the single data points but on an aggregated level. For example in a cube containing turnover data, only the trend over the years (time dimension) or the variations between the countries (level country in a hierarchy city→country→continent within the region dimension) is of

[3] http://msdn.microsoft.com/de-de/library/ms717005.aspx

interest. To summarize the single data points to an aggregated value, some conditions (so called summarizability) must be fulfilled and regulations are needed that prescribe the mapping of many single values to one aggregated value (so called aggregation functions) [NiNi10].

In common data warehouses, the following standard aggregation functions are supported when executing operations on an OLAP cube:

- Minimum
- Maximum
- Sum
- Average
- Range
- (Distinct) Count

These and all other aggregation functions can be classified in the following three types, describing if and how the already aggregated values can further be aggregated to a higher level [LeTh09]:

- Distributive:
  These aggregation functions can be defined as a structural recursion schema, which means that a further aggregation in a higher level can be done without any other information, e.g. minimum. Thus, for aggregates only a fraction of the original storage space is required. The calculation of further aggregated values only requires the current aggregates, which results in a good performance as compared to the other types of aggregation functions.
- Algebraic:
  These aggregation functions can be expressed by finite bounded algebraic expressions, which means that a further aggregation in a higher level can be done, but additional information is required, e.g. average, where count is the additional information. Thus, for aggregates less storage space than for the original data is required but more than for aggregates with distributive aggregation functions. The calculation of further aggregated values also only

requires the current aggregates, which results in a good performance as compared to the other types of aggregation functions, too.

- Holistic:

  These aggregation functions are not bound on the size of storage, needed to describe the aggregates, which means that a further aggregation is only possible on the original data, e.g. median. Thus the storage space for aggregates is not less than the storage space for the original data. The calculation of further aggregated values requires the original data, which results in a bad performance as compared to the other types of aggregation functions.

Summarizability is the correctness of aggregation operations, which means that a function computed from the aggregated data is the same as it would be computed from the original, not aggregated data. The following three conditions must be fulfilled to guarantee summarizability [LeSh97]:

- Disjointness:

  This rule means that a dimension member may belong to only one parent member.
- Completeness:
  - The first part of this rule means that each member of the higher levels must have child members till to the lowest level of the hierarchy.
  - The second part of this rule means that each member must belong to a particular parent member.
- Consistency:

  This rule means that it depends on the interaction of the following three characteristics whether summarizability holds:
  - Type of the dimension:
    - Temporal
    - Non-temporal
  - Type of the measure:
    - Stock (measured at a particular point of time, e.g. number of citizens)
    - Flow (refer to periods and are recorded at the end of these periods, e.g. annual income)

- Value-per-unit (determined for a fixed time, e.g. item price)

- Statistical aggregation function associated (described above)

For example, it does not make sense to summarize (aggregation function: sum) the number of citizens (type of measure: stock) over the years (type of dimension: temporal). The following table gives an overview of the combinations that guarantee summarizability for temporal and non-temporal dimensions, if disjointness and completeness are given.

	Temporal			Non-temporal		
	Stock	**Flow**	**Value-per-unit**	**Stock**	**Flow**	**Value-per-unit**
Min	ok	ok	ok	ok	ok	ok
Max	ok	ok	ok	ok	ok	ok
Sum	not ok	ok	not ok	ok	ok	not ok
Avg	ok	ok	ok	ok	ok	ok
Range	ok	ok	ok	ok	ok	ok

Table 1: Consistency [modified from LeSh97]

2.1.2 **Business Intelligence**

The term Business Intelligence (BI) was mint by the Gartner group with the following statement: ‘Data analysis, reporting, and query tools can help business users wade through a sea of data to synthesize valuable information from it – today these tools collectively fall into a category called ´Business Intelligence´ ‘[AnAS04, p.18 et seq.].
BI does not denote a specific system or application; it can rather be understood as a conglomeration of concepts and technologies to support the decision process. Kemper et. al. define Business Intelligence as an integrated, company-specific, IT-based, entire approach to support operational decision-making [KeMU06, p.8].

2.1.2.1 BI process

From the process-driven perspective, BI also can be regarded as a process from different and distributed data up to specific knowledge. Therefore, first the relevant data has to be selected and a data preprocessing is necessary. After a transformation, the data has to be provided in OLAP-Cubes, where different analysis methods finally generate specific knowledge after interpretation of the results [Humm08]. The following figure shows such a generic BI process.

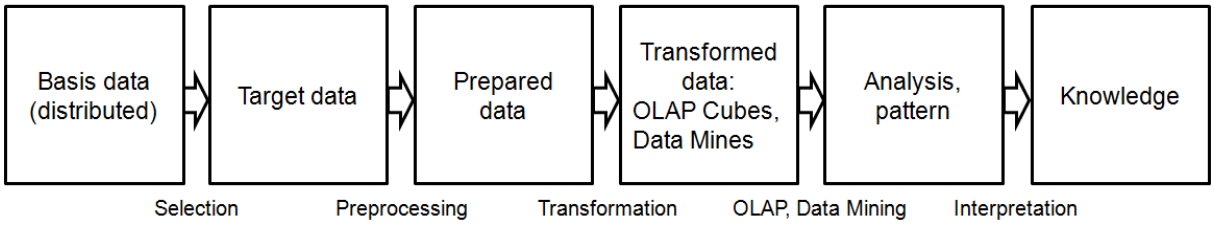

Figure 2: A generic Business Intelligence process [modified from Humm08]

The generation of OLAP cubes and hierarchies out of Statistical Linked Data is a concretization of a part of this generic BI process. Linked Data can be understood as distributed basis data and the above steps can take place till OLAP cubes are generated. On these, different analysis can be made that were finally interpreted to knowledge.

2.1.2.2 Facets of BI

The technologies and concepts, which are suitable to support decision making can be arranged in a two-dimensional illustration. At this, the vertical axis shows the phases from data preparation to data analysis. The horizontal axis shows the focus from technology to application.

This illustration clarifies the differentiation of BI in different perspectives. Since BI in the broad sense includes all technologies and concepts to support decision-making, the analysis-oriented understanding of BI concludes all applications, in which a user has a direct and interactive access to the system. BI in a strict sense only includes applications that immediately support decision making. Since OLAP is a core application of BI in a strict sense, it is an essential technology to support decision making.

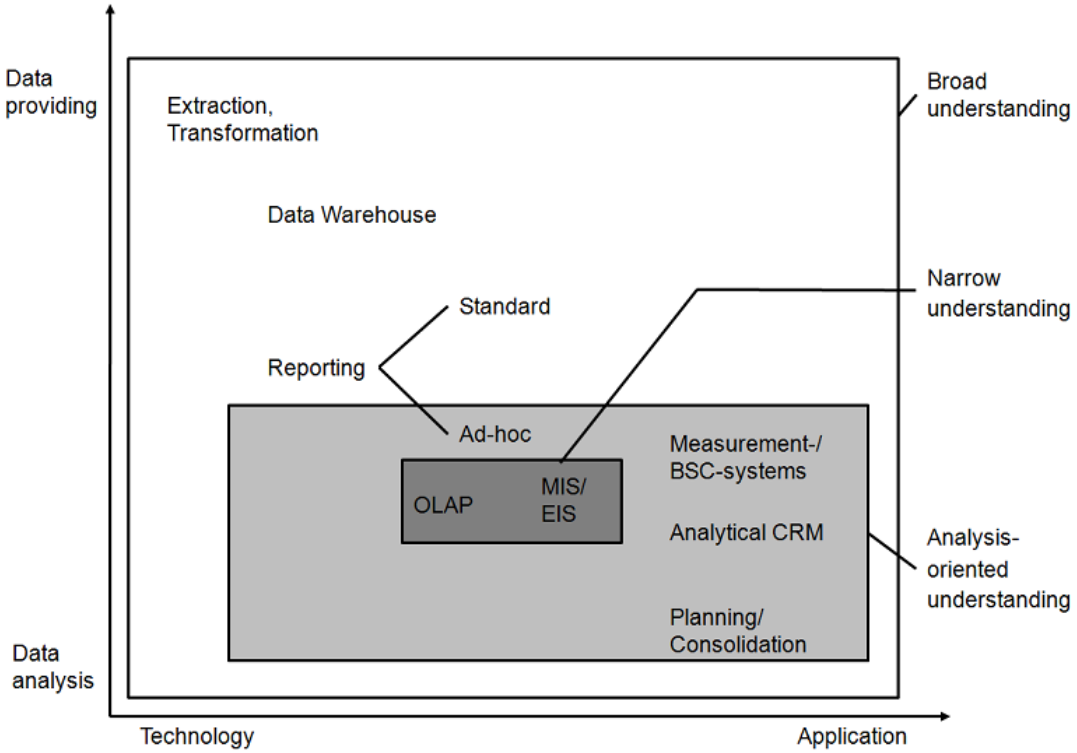

Figure 3: Facets of Business Intelligence [KeMU06, p.4]

2.1.2.3 Data – Information - Knowledge

From the process-oriented perspective, BI can be regarded to generate knowledge out of data. Thus, data is transformed to information, which is an intermediate step and then transformed to knowledge, which is used to make decisions.

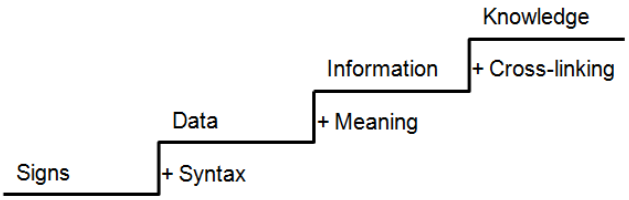

Figure 4: Stairs of knowledge [modified from Nort05, p.32]

Data, constructed out of signs on the basis of certain principles, is the basic module of information science and represent facts.

By adding a meaning and a purpose to data, it is transformed to information and the data becomes dependent of the context, in which it is used.

Knowledge is formed by evaluating, comparing or combining information. This implicates that knowledge is always committed to persons or organizations, where it can be used to make useful and responsible decisions. Hence, a hierarchy also illustrates an effective form of knowledge representation, because prior domain knowledge relevant to data is encoded [PoRa99].

2.1.3 **Data Warehouse**

To prepare the data for analysis purposes such as OLAP, a special system, which serves as data pool, is required. This special database is called data warehouse, where all data and metadata can be stored for analysis and archiving purposes.

One of the first definitions points up the significant properties: ‘A data warehouse is a subject oriented, integrated, non-volatile, and time variant collection of data in support of management’s decisions’ [Inmo02, p.31]. These four properties, which all are relevant for decision making, are explained below:

- Subject orientation: The focus of the system does not concentrate on processes but on the modeling of a subject.
- Integration: There is integrated data existent, which may originate from several and different sources.
- Non-volatility: The data is stored persistently, thus the data is not overwritten.
- Time-Variant: The data can be analyzed within the time dimension.

Considering these aspects, data warehouses elementary distinguish from transactional databases in the following main characteristics [BaGü04, pp. 9-11]:

- Query-Performance: Data warehouses are optimized for long, large and complex read accesses instead of short, small, easy and single-tuple read and write accesses.
- Historical data: Decision support in data warehouses requires consolidated and historical data from different and heterogeneous sources instead of only the current data in one transactional database.
- User: Data warehouses are designed for a few data analysts, manager and controller instead of a large number of case workers.

In consequence of all these reasons, data warehouses are implemented separately from operational databases, so that queries use the data pool of the data warehouse instead of calculating the result on-the-fly by accessing the data sources of the data warehouse in the running time of the query. To have always new data available, data warehouses have to be updated in a pre-defined rhythm, e.g. every night, monthly. This is done by the ETL process.

2.1.3.1 ETL process

To have data for the OLAP operations in a data warehouse available, they have to be extracted from the data sources, transformed for analysis purposes and saved in the data warehouse. This recurring procedure is called ETL-process and consists of the following steps [BaGü04, p. 81 et seqq.]:

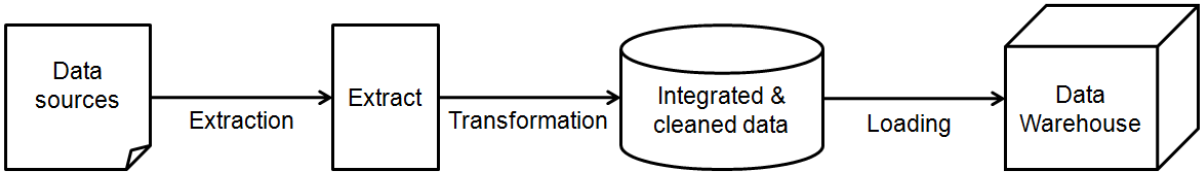

Figure 5: ETL-process

- Extraction: The first step is the extraction of the data from the sources, which can be done in miscellaneous manners (e.g. periodic, event-triggered, query-triggered). The outcome of this step is the transferring of the relevant data to a work area, where it can be transformed.
- Transformation: The second step is the transformation of the data, which means to adapt data, schema and data quality to the requirements of the end users. Mainly, the heterogenic data has to be integrated (e.g. adapting data types, harmonizing strings) and cleaned (e.g. avoiding redundancy, checking for consistency)
- Loading: The last step is the loading of the data into the data base, where it is stored persistently. After one initial load, the OLAP cubes are updated and the data is available for analysis.

This master thesis attaches importance to the ETL-process, since Linked Data is extracted from the web and transformed that it is adequate to the OLAP4J standard, whereby a special focus is on hierarchies. Besides the possibility to directly use Linked Data hierarchies for example with OLAP4J, which is described in section 4 ‘Transforming Linked Data into OLAP hierarchies’, the constructed hierarchies could also be further transformed and loaded in a data warehouse, where they can be used.

2.1.3.2 Star and snowflake schema

On the conceptual level, modeling the data is done in multidimensional cubes. The conversion of the semantic, conceptual level to the logical, intern level of the database he may occur in a multidimensional way, which is called MOLAP and in a relational way, which is called ROLAP [BaGü04, p. 201 et seqq.]. Since the advantages of ROLAP are open standards and scalability, the advantages of MOLAP are direct implementation of the conceptual level of OLAP, intuitive operating and analytical powerfulness [BaGü04, p. 242 et seqq.]. To combine the advantages of ROLAP and MOLAP, the conversion can also be done in a hybrid way, which is called HOLAP.

Because relational databases are popular in theory and practice, ROLAP is explained in more detail, which can be further divided in the star and snowflake schema:

- Star schema:
  The star schema consists of one fact table and several dimension tables. Since the measures are contained in the fact table, the dimension tables contain the attributes of the dimensions. Each dimension table is connected with its primary key with the fact table, thus the total of all foreign keys referencing to the fact table is its primary key. If a dimension has a hierarchical structure, the star schema is denormalized, since there are functional dependencies between the non-key attributes representing the hierarchy.

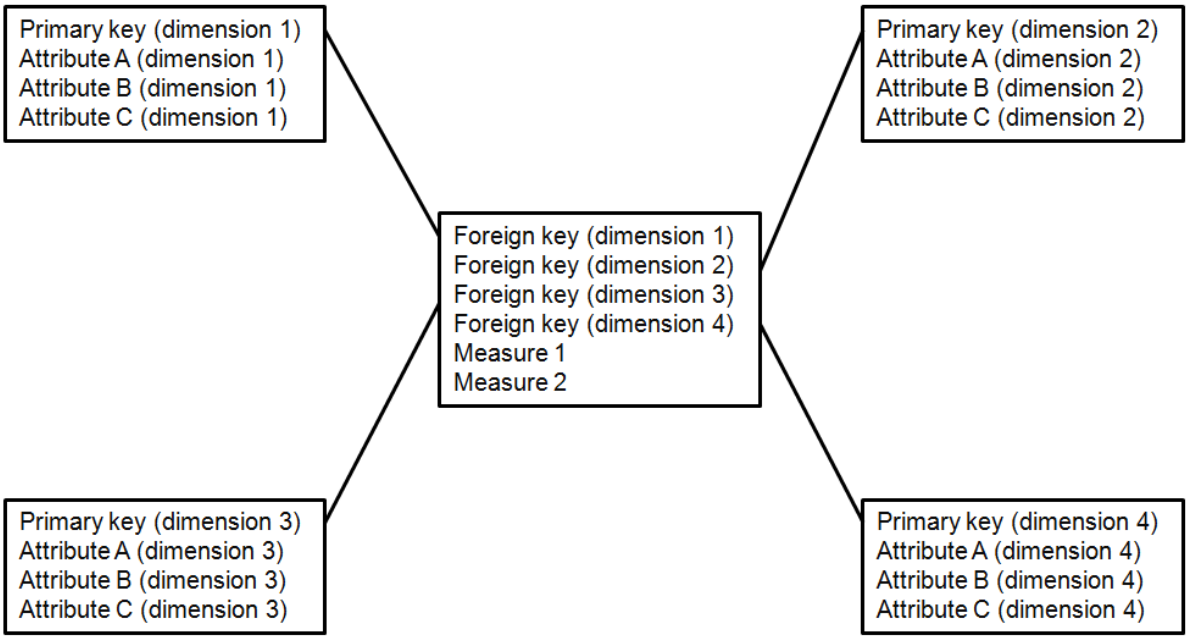

Figure 6: Star schema

- Snowflake schema:

  The snowflake schema also consists of one fact table, which contains the measures and several dimension tables. In contrast to the star schema, the snowflake schema is normalized. Thus, one dimension covers several tables for the hierarchy levels. The primary key of the lowest level of each hierarchy is referencing to the fact table, where the total of all foreign keys referencing to the fact table is its primary key.

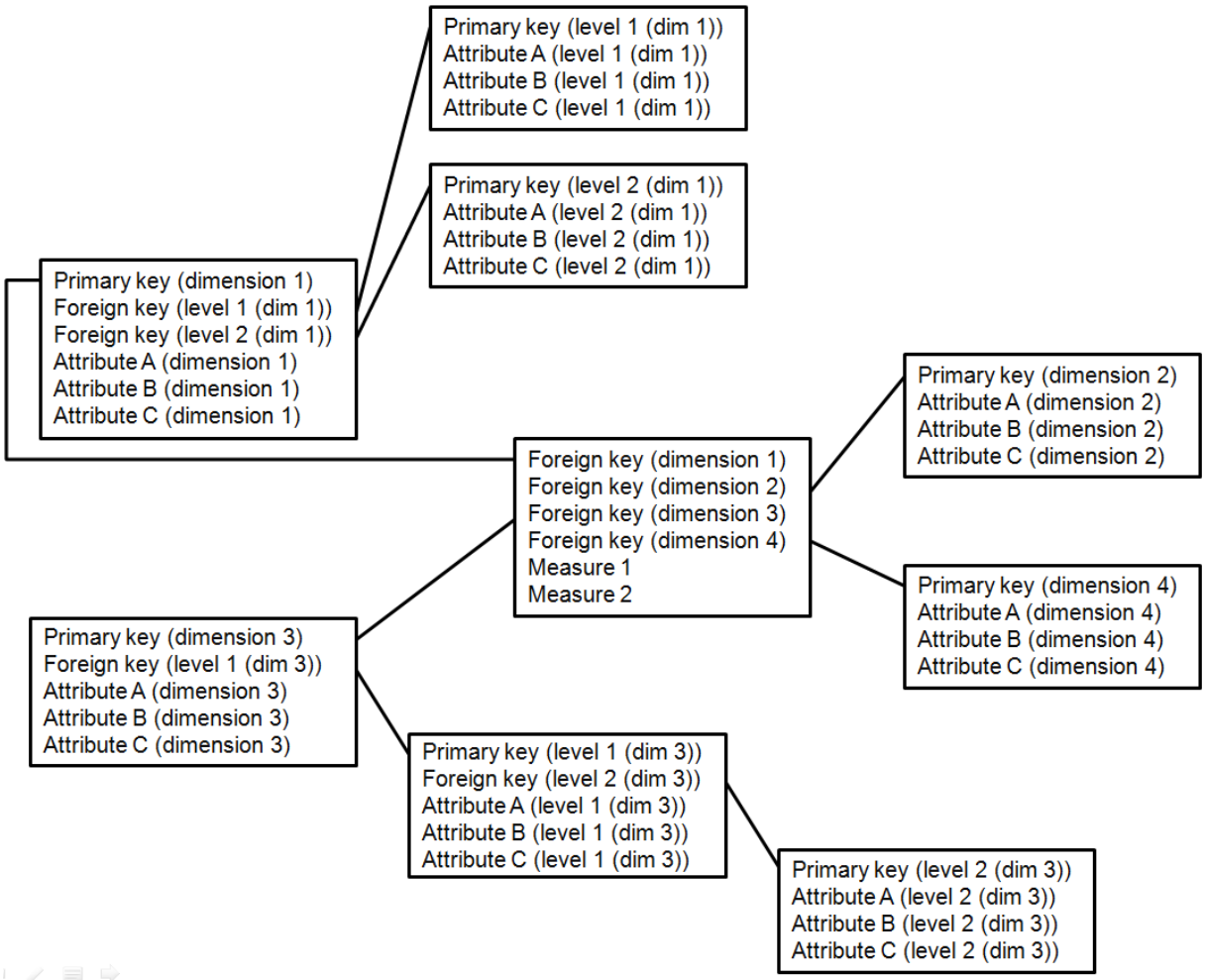

Figure 7: Snowflake schema

2.2 Linked Data

Linked Data is an approach to connect data from different sources on the World Wide Web by creating typed links between different data sets [BiHB09]. If the data is freely available, the term Linked Open Data is used. Linked Data is based on the Semantic Web standards, whereby the unique identification of each thing, such as metadata elements or certain entities is very important [ZaHM11]. Therefore Uniform Resource Identifiers (URIs) are used. Tim Berners-Lee provided the four principles for Linked Data [Bern06]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up an URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs, so that they can discover more things.

The publication of data as Linked Data enables advantages for all, data providers, developers and end-users [ZaHM11]. Data providers have the possibility to enrich their own data by linking to other data sets on the web. Developers benefit from Linked Data also, because they are not restricted on one data set and can easily integrate data from other sources. Finally, both aspects are also relevant for end-users, because they have useful applications and many data sets available.

2.2.1 **RDF, RDFS and OWL**

Resource Description Framework (RDF)[4] is a formal language for representing structured information about things on the World Wide Web [HiKR09, pp.19]. The intention is not to display data correctly, but rather re-combination of information contained in it.

The conceptual representation of an RDF document is a set of nodes, which are linked by directed edges, so that a graph is formed. Each statement is structured by subject-predicate-object sentences, whereby all parts consist of an URI. A statement is also called a triple [HiKR09, pp. 20-25].

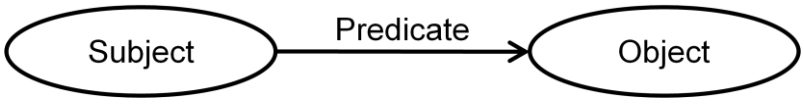

Figure 8: Triple consisting of subject, predicate and object

[4] http://www.w3.org/TR/rdf-primer/

There are several serialization formats of RDF. Since many programming languages support Extensible Markup Language (XML)[5] libraries, the XML based serialization RDF/XML is a common practice. XML itself is a markup language, intended to use for data exchange and electronic publishing. To define structures for XML documents, XML Schema[6] can be used. Both, XML and XML Schema are W3C Recommendations [HiKR09, pp. 353-361].

However, this master thesis uses the Turtle representation of RDF, because it can more easily be accessed by humans. A triple is denoted with turtle like this:

ex:subject	ex:predicate	ex:object

The part before the double dot is an abbreviation for a longer part. This concept is called namespaces. There may be different abbreviations for different longer parts. A list of all relevant namespaces for this master thesis can be found in the appendix. The abbreviation ex: always stands for an example namespace.

RDF can semantically be enriched by using Resource Description Framework Schema (RDFS)[7]. This also graph-based language provides the possibility to generate simple ontologies, so that knowledge about a domain of interest can be represented in a standardized way [HiKR09, pp. 46-67].

Resources can be typed, i.e. to mark them as elements of a certain aggregation. The individual elements are called instances; a set of resources is called class. Classes can be sub-classes of other classes, so that a class hierarchy is generated. This hierarchical relationship is used in one approach of this master thesis to generate OLAP hierarchies.

Predicates in a triple are also called properties, since they describe a relationship between two other resources (subject and object). Also properties can be sub properties of other properties, so that a property hierarchy is generated. Furthermore, properties can be restricted, by setting limits for possible classes of subject and object.

[5] http://www.w3.org/XML/
[6] http://www.w3.org/XML/Schema
[7] http://www.w3.org/TR/rdf-primer/#rdfschema

Considering RDF and RDFS, we have to distinguish assertional knowledge, which is represented in RDF and makes propositions about concrete entities and terminological knowledge, which is represented in RDFS and gives background information about the domain of interest [HiKR09, p. 66]. Both, RDF and RDFS are W3C Recommendations.

Since RDF and RDFS provide only very limited expressive means, the Web Ontology Language (OWL)[8] is used, to represent more complex knowledge [HiKR09, p. 111]. This language is also a W3C Recommendation and has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. The basic buildings of OWL are classes, roles and individuals [HiKR09, pp. 111-158]. Individuals are instances of classes and classes are the same like in RDFS. In contrast to RDFS, OWL properties are called roles. There are abstract roles, which link two classes and concrete roles, which link two individuals. The essential contrasts to RDFS are the advanced concepts class restrictions and role restrictions. With these concepts, it is possible to express a stronger semantic meaning than in RDFS.

2.2.2 **RDF Data Cube Vocabulary and SDMX**

The RDF Data Cube Vocabulary is an approach to publish multi-dimensional data on the web [CyRT10]. It is built upon the Statistical Data and Metadata eXchange (SDMX), which is the ISO-standard (ISO/TS 17369:2005) for statistical data exchange. The RDF Data Cube Vocabulary is an OWL ontology and compatible to Linked Data, which enables the following advantages:

- Observations become web-addressable to be annotated or linked to.
- Data between statistical and non-statistical data sets can be combined.
- Flexible, non-proprietary and machine readable means of publication.
- Standardized tools and components can be reused.

The following figure gives an outline of the RDF Data Cube Vocabulary:

[8] http://www.w3.org/TR/owl-features/

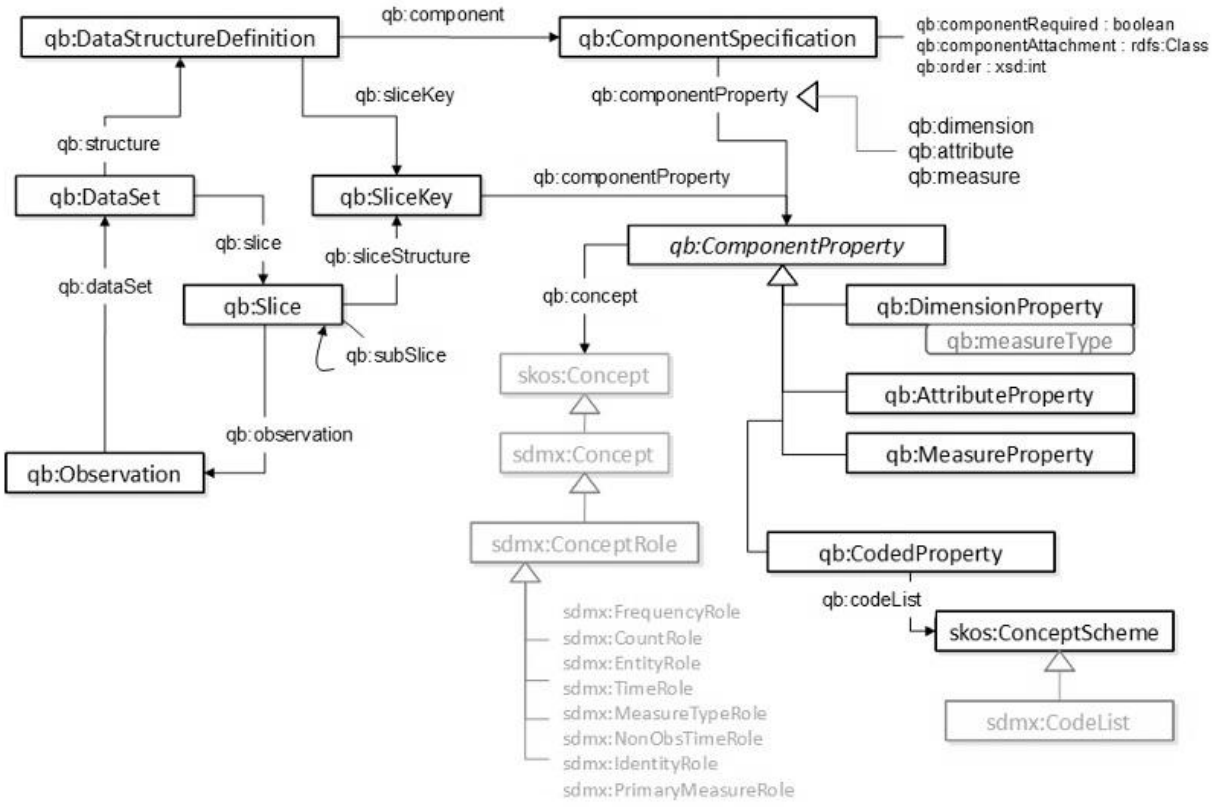

Figure 9: RDF Data Cube vocabulary [CyRT10]

A cube is represented by qb:DataSet and is linked by qb:structure to a qb:DataStructureDefinition. The qb:DataStructureDefinition defines the structure of one or more cubes, which consists of a set of components, which are represented by qb:ComponentSpecification. There is the possibility to qualify a qb:ComponentSpecification by the following:

- qb:componentRequired to set a component optional.
- qb:order to order the components for user interfaces.
- qb:componentAttachement to attach attributes at other levels of the structure.

Furthermore, each qb:ComponentSpecification refers by the qb:ComponentProperty to one of three kinds of components, which are subclasses of qb:ComponentProperty:

- qb:DimensionProperty: To identify the observations, e.g. time, region.
- qb:AttributeProperty: To qualify and interpret the observed value(s), e.g. unit, scaling factor.
- Qb:MeasureProperty: The Phenomenon being observed, e.g. turnover, costs.

Each qb:ComponentProperty refers by the qb:concept to a skos:Concept, a predefinied terminology to support interoperability and comparability. An observation, to which all corresponding dimensions, measurements, attributes are attached, is typed as qb:Observation and is linked by the qb:dataSet to the qb:DataSet.

The RDF Data Cube Vocabulary also provides the possibility to define slices, which allow to group subsets of observations together. Since a slice is also a result of an OLAP operation on a multidimensional cube, a defined slice in the RDF Data Cube Vocabulary is not relevant for this master thesis.

To define members of a dimension unambiguously, a qb:DimensionProperty can be linked via rdfs:range to a certain data type or via qb:codeList to a skos:ConceptScheme. By using a skos:ConceptScheme and other parts of the Simple Knowledge Organization System (SKOS)[9], hierarchical structures between members can be expressed. This vocabulary is described and critically validated in section 3 ‘Expressing OLAP hierarchies in RDF‘.

2.2.3 **SPARQL**

The SPARQL Protocol And RDF Query Language (SPARQL)[10] is a W3C Recommendation for querying RDF based information and for representing the results. The core of it are simple graph patterns, which can be extended with advanced query patterns, such as filtering, grouping, alternatives. Although the syntax and usage of SPARQL is similar to the query language for relational databases Structured Query Language (SQL), it must be noticed that the two languages operate on very different data structures [HiKR09, p. 262].

[9] http://www.w3.org/TR/2009/REC-skos-reference-20090818/
[10] http://www.w3.org/TR/rdf-sparql-query/

Since the official W3C Recommendation of SPARQL is version 1.0, there is already a W3C working draft for SPARQL version 1.1[11] with extended features, which is used in this master thesis to query RDF data.

2.2.4 **Semantic Web**

Linked Data as approach to connect data on the web is a special aspect of the broader Semantic Web [Sack10, pp. 17-18]. The Semantic Web itself is defined as an extension of the current Web, in which information is given a well-defined meaning, better enabling computers and people to work in cooperation [BeHL01]. This definition intonates that the meaning and use of information is essential for the collaboration over the web. Many aspects, such as the stronger collaboration in scientific progresses (data interchange) or the interconnection of workflows and business processes because of increased cost pressure and competition in business are fueling the effort of the Semantic Web [KaBM08, pp. 4-5].

From the technology-oriented perspective, the Semantic Web consists of different standards that are based on each other. They can be arranged in a Semantic Web Layer Cake [Brat07]. For this master thesis, the layers above OWL are not relevant.

[11] http://www.w3.org/TR/sparql11-query/

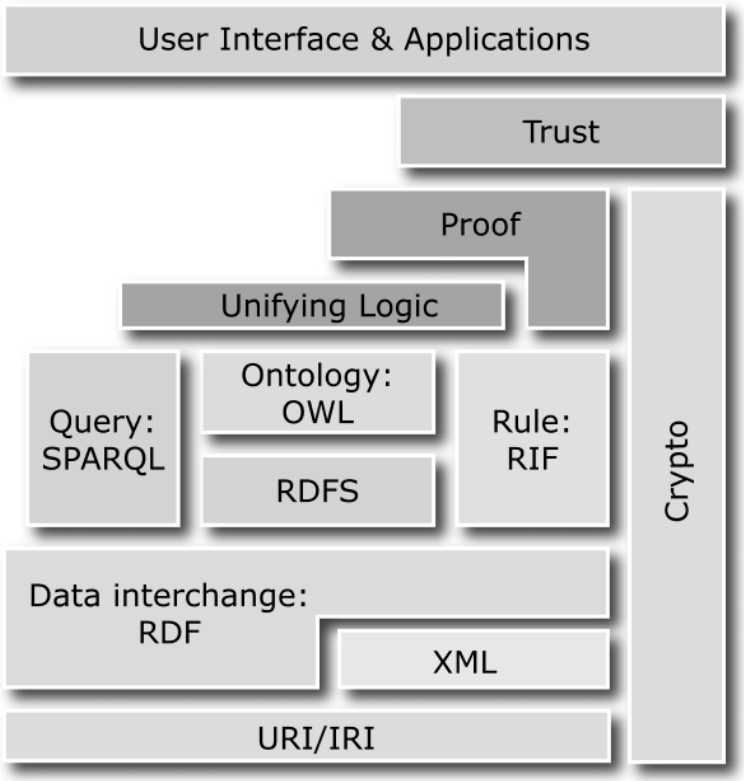

Figure 10: Semantic Web layer cake[12]

2.2.5 Linking Open Data Cloud and Statistical Linked Data sets

As described above, the main idea of Linked Data is to interconnect different data sources on the World Wide Web by creating typed links between different data sets. All data sets, following the four Linked Data principles, which are connected to each other, can be taken together in a cloud diagram, which is updated periodically if additional data sets are available [CyJe11]. Within this cloud diagram, also Statistical Linked Data sets can be found.

[12] http://www.w3.org/2007/03/layerCake.png

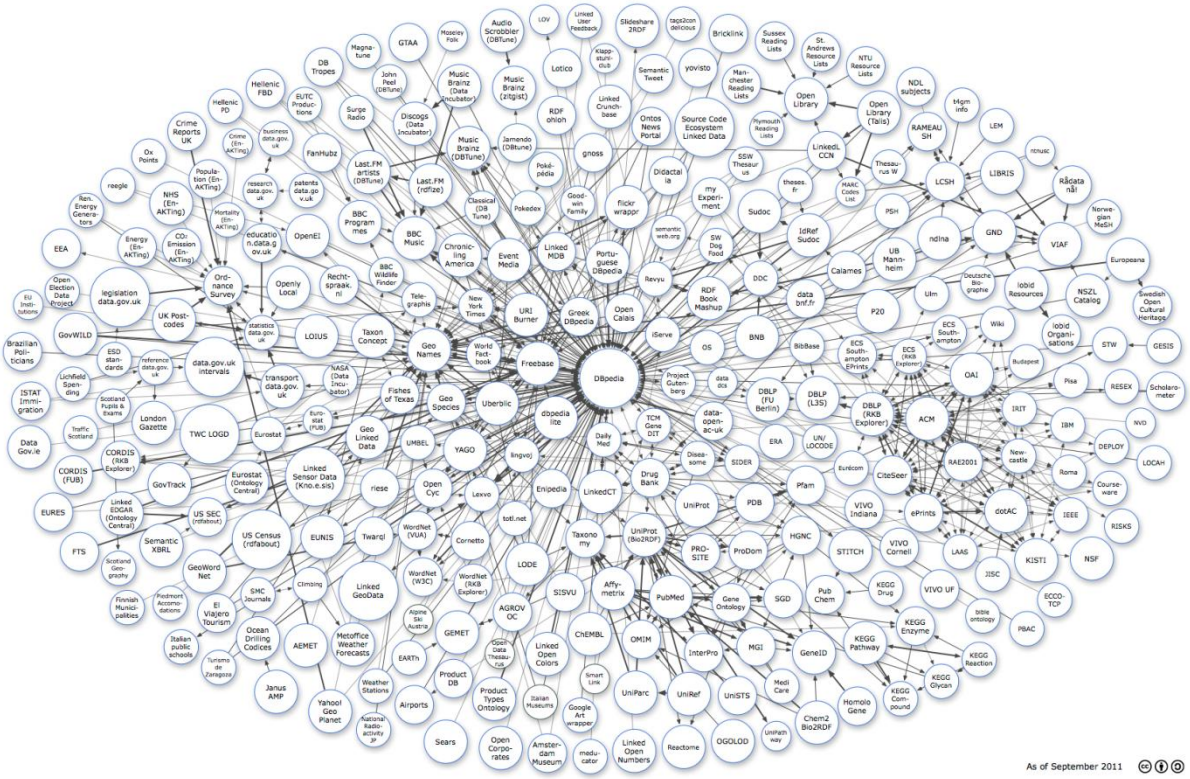

Figure 11: Linking Open Data cloud diagram, September 2011 [CyJe11]

Since it is the aim of this master thesis to derive hierarchies out of Statistical Linked Data, relevant data sets are searched. The following assumptions are made that a data set can be used to test and evaluate the developed approaches:

- The data set follows the four Linked Data principles.
- The data set includes Statistical Linked Data.
- The data set has a correct and corresponding qb:DataStructureDefinition from the RDF Data Cube Vocabulary.

The following table includes some examples of data sets that were found on the web and fulfill all assumptions. The full table, including links to the data sets and data structure definitions has also been published. [13]

[13] http://planet-data-wiki.sti2.at/web/Datasets

Topic/Example	Publisher	Measures	Dimensions
electoral statistics of a German state[14]	GESIS – Leibniz Institute for the Social Sciences	obsValue	geo party
financial data for the UK government (COINS)[15]	data.gov.uk	amount	refPeriod dataType dataSubtype departmentCode accountCode programmeObjectCode counterpartyCode
Real GDP growth rate in Europe[16]	Eurostat	obsValue	date unit geo timeFormat freq obsStatus
data of companies, required by law to file forms(EDGAR)[17]	U.S. Securities and Exchange Commission	accountsPayableCurrent accruedLiabilitiesCurrent (…)	issuer date segment
payments to suppliers[18]	Lichfield District Council	netAmount	reference payer payee date expenditureLine expenditureCategory payment

Table 2: Examples of data sets using the RDF Data Cube Vocabulary

[14] http://gesis-lod.appspot.com/
[15] http://data.gov.uk/resources/coins
[16] http://estatwrap.ontologycentral.com/page/tsieb020
[17] http://edgarwrap.ontologycentral.com/
[18] http://spending.lichfielddc.gov.uk/

2.2.6 **Linked Data as a source for data warehouses**

The concept of this master thesis is a concretization of the generic BI process, since Statistical Linked Data is transformed to an OLAP cube, whereby hierarchies are generated.

There are several general advantages of using ontologies for the design of data warehouses, because ontological knowledge may enrich a multidimensional model, e.g. to find hierarchies [PaMa11]. A concrete ontology is the RDF Data Cube Vocabulary, including Statistical Linked Data. If this serves as data source for data warehouses, the following advantages can be recognized:

- Additional method to analyze Linked Data:
  Besides the possibility to browse Linked Data with a Semantic Web Browser or a Faceted Search Browser, it can now be viewed with the common OLAP operations in different perspectives and on different levels of detail.
- Additional type of data sources for data warehouses:
  Data warehouses can be filled with Linked Data, which is an additional type of a data source. So not only other data, for example out of relational databases or other web data, for example published in XML, can be integrated.
- Integration of Linked Data and OLAP cubes:
  Statistical Linked Data, both external and captive, can be transformed to a cube in a data warehouse, where it can be combined with other data, for example captive data out of customer relationship (CRM) or Enterprise Resource Planning (ERP) systems in already existing OLAP cubes.
- Additional semantics:
  The additional semantics of the Linked Data can be used to enrich the data or metadata of the existing data warehouse.

Summarizing these aspects it can be noticed that the two concepts Linked Data and OLAP profit by each other.

2.3 Hierarchies

A hierarchy is a system of elements, which are subordinate or superordinate to each other. The need for hierarchies in a dimension of a cube is, because things in real world can also exhibit hierarchical structures. Because of this special relationship, the elements of dimensions, called members, can be arranged in one or more hierarchies.

Since a subordinate member is called child, a superordinate member is called parent [MaZi04]. Dependent on the children and parents, each member is assigned to a certain level. The sequence of levels is called path and the number of levels, which are forming a path is called path length. The uppermost level, having no parent level, is called root and levels having no children are called leaf levels. The minimum and maximum numbers of members in one level which are related to a member of another level is indicated by the cardinality. This master thesis completes these terms, used in [MaZi04], by the terms root distance and stage. The number of levels without skipping an intermediate level between the root and a particular level is called root distance of this level and all levels that have the same distance to the root are located on the same stage.

In the context of OLAP, a formal definition for a hierarchy is as follows: “A hierarchy is a set of variables which represent different levels of aggregation of the same dimension and which are linked between them by a mapping” [PoRa99]. This means, a hierarchy shows the relationships between domains of values (variables), called levels and the variable instances, called members.

Hierarchies enable both, navigation paths through a dimension for end-users and aggregation paths for the associated measures [GaGP09, p.58]. Going along a hierarchy, drill-down operations towards a lower, more detailed level and roll-up operations towards a higher, more generalized level are possible. This master thesis introduces the restriction that there are no loops within a path, meaning that a roll up respectively drill down operation never links back to a lower, more detailed respectively higher, more generalized level.

From the view of mathematics, these operations are transformations of values from one domain to values of another smaller or bigger domain, whereby a mapping

between the domains is used [PoRa99]. If the summarizability conditions of disjointness and completeness are fulfilled, the mapping defines a containment function and it is called full mapping. If there is a full mapping between each adjacent couple of variables, the hierarchy is called classification hierarchy, otherwise aggregation hierarchy.

A proposal for a formal notation of hierarchies in a multidimensional model can be found in [Vass98]. More than one parent member in a higher level (non-strict hierarchies) and descriptive attributes of a level are not expressible with this notation. For this reason the following table gives an overview of the already developed notation and a proposal for an extension to express all required elements of a proper hierarchy. It has to be noticed that the formal defined operations, in particular level_climbing and function_application in [Vass98] also have to be adapted, since the notation is extended. Because the correct aggregation is not part of this master thesis, these cube operations are not defined formally.

Element	Explanation	Notation
Notation proposed by [Vass98]:		
Dimension		D_i
Level		DL_i
Levels of a dimension		$levels(D_i)$
Hierarchy	Lattice of levels	$(H, \leq)$
Dimension path	Linear, totally ordered list of levels	D_{pi}
Levels of a dimension path		$levels(D_{pi})$
Paths of a dimension		$paths(D_i)$
Member		x
Members of a level		$dom(DL_i)$
Operator h	Assigning levels to dimensions	$h(DL_i) = D \text{ if } DL_i \in levels(D)$
Function level	DL_i is the k-th level of dimension path D_{pi} with DL_0 denoting the lowest level	$level(DL_i) = k, \text{ if } DL_i = levels(D_{pi})[k]$
Children of a member	Relationship to members on a lower level	$descendants(x, DL) = \{x_1, x_2 \ldots, x_k\}, x_1, x_2 \ldots, x_k \in dom(DL), DL < DL_0$
Proposed extension:		
Parents of a member	Relationship to members on a higher level	$ancestors(x, DL) = \{y_1, y_2 \ldots, y_k\}, y_1, y_2 \ldots, y_k \in dom(DL), DL_0 < DL$
Descriptive Attribute		a
Descriptive Attributes of a level		$attributes(DL_i)$
Characteristic of attributes		$characteristic(x, a, DL_i)$

Table 3: Formal notation for a hierarchy

According to the proposal of [MaZi05], hierarchies in a multidimensional model can also be classified by different criteria, which are described in the following subsections. The following figure gives an overview, including the relationships of the criteria:

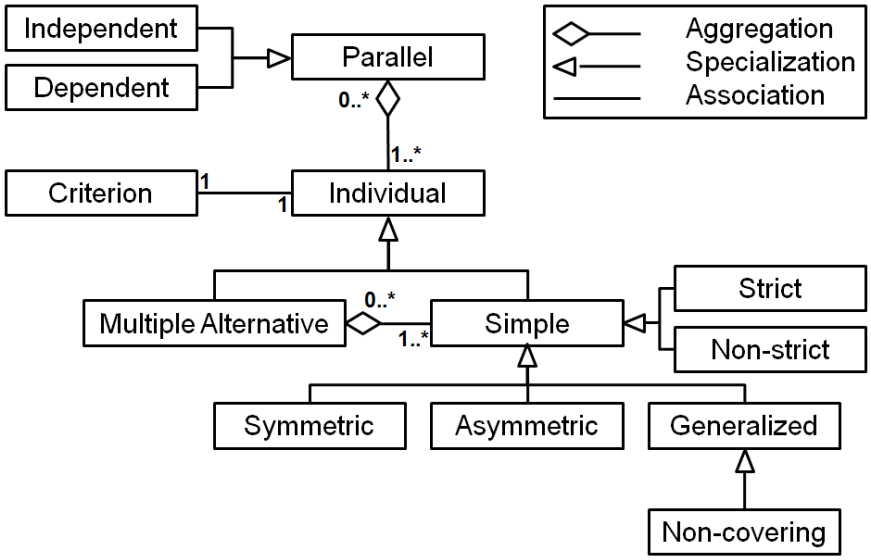

Figure 12: Metamodel of hierarchy classification

A graphical example both on the schema level, which describes the relationships of the levels, and instance level, where concrete instances are populated, is given for each type. Therefore the following notation is used:

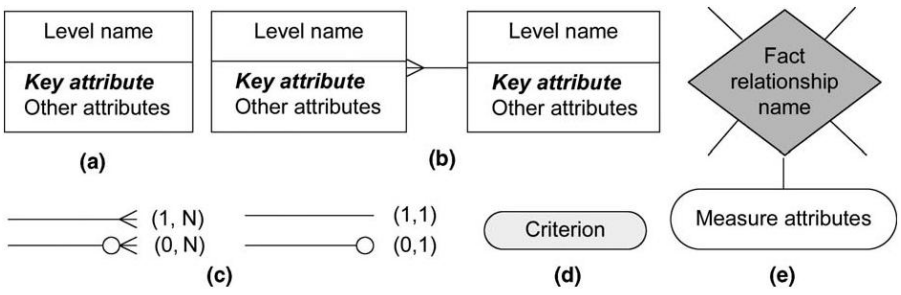

Figure 13: Notation of hierarchies in a multidimensional model: (a) level, (b) hierarchy, (c) cardinalities, (d) analysis criterion, and (e) fact relationship

The main distinction of different hierarchies is substantiated in the number of analysis criteria within one dimension. An analysis criterion represents a substantially criterion

used in analysis. For example the regional dimension can have the two analysis criteria geographical location and organizational structure. So we fundamentally differentiate individual and parallel hierarchies.

2.3.1 **Individual hierarchies**

In an individual hierarchy, only one criterion for analysis is used. There may be one to n paths through the hierarchy, but finally all paths end at the schema level on the same level. So the distinction between simple and multiple alternative hierarchies is caused in the number of paths through the hierarchy at the schema level.

2.3.1.1 Simple hierarchies

A simple hierarchy is a specialization of an individual hierarchy, where exactly one path at the schema level is possible. It can be represented as a tree and it is characterized by two properties, namely type and strictness.

2.3.1.1.1 Type of simple hierarchies

2.3.1.1.1.1 Symmetric hierarchies

In a symmetric hierarchy, all levels are mandatory, so that all branches have the same length. At the instance level, each parent member must at least have one child member.

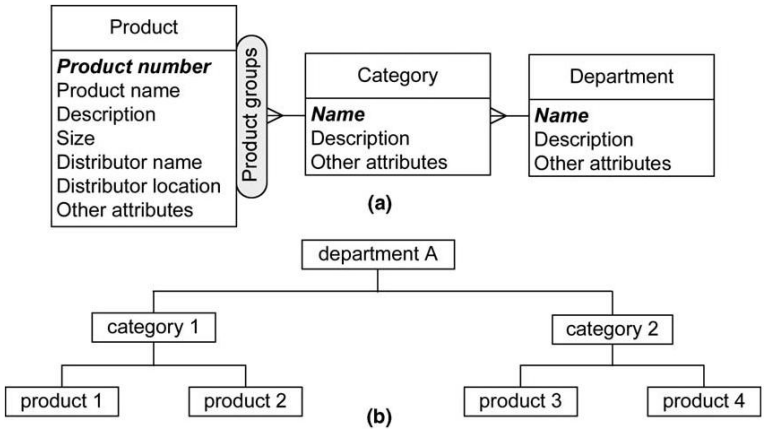

Figure 14: Example for a symmetric hierarchy: (a) model and (b) instances

2.3.1.1.1.2 Asymmetric hierarchies

In contrast to symmetric hierarchies, not all levels are mandatory in an asymmetric hierarchy. This implicates that the branches do not have the same length so that this type can be represented as an unbalanced tree. A child member exactly belongs to one parent member.

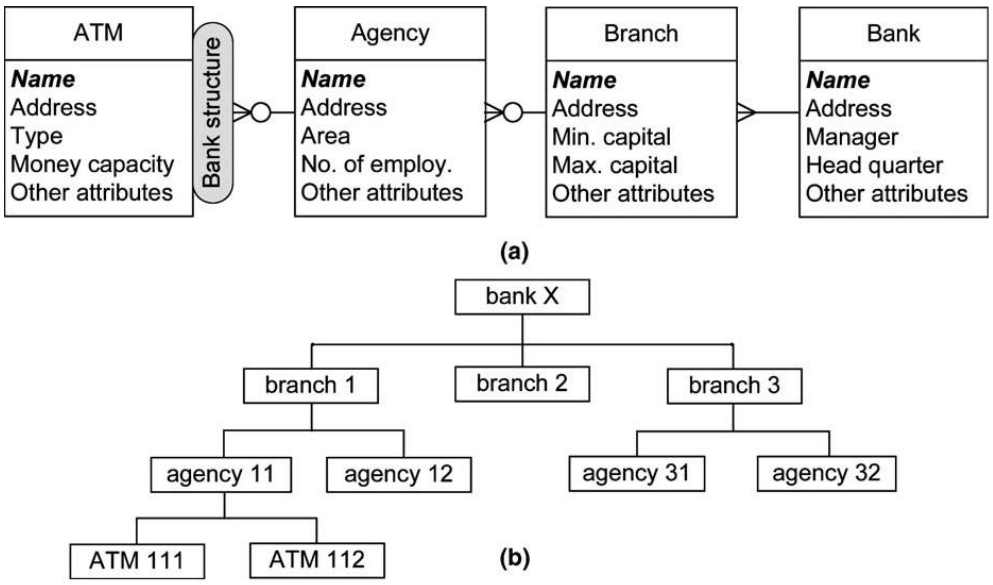

Figure 15: Example for an asymmetric hierarchy: (a) schema and (b) instances

For this hierarchy, completeness is not given, so that the summarizability condition is not fulfilled. Therefore two alternative solutions are provided:

- Transforming an asymmetric hierarchy into a symmetric one using placeholders
- Creating parent child relations

2.3.1.1.1.3 Generalized hierarchies

A generalized hierarchy represents a generalization/specialization relationship. This concept is also used in object oriented programming, where it is called inheritance. There are two types of levels: Common levels, where the members within one level have the same attributes and specific levels, where the members within one level have different attributes. At the schema level, multiple exclusive paths that share some levels are possible and at the instance level a member exactly belongs to one

path. The levels where the paths are split respectively joined are called splitting or joining level.

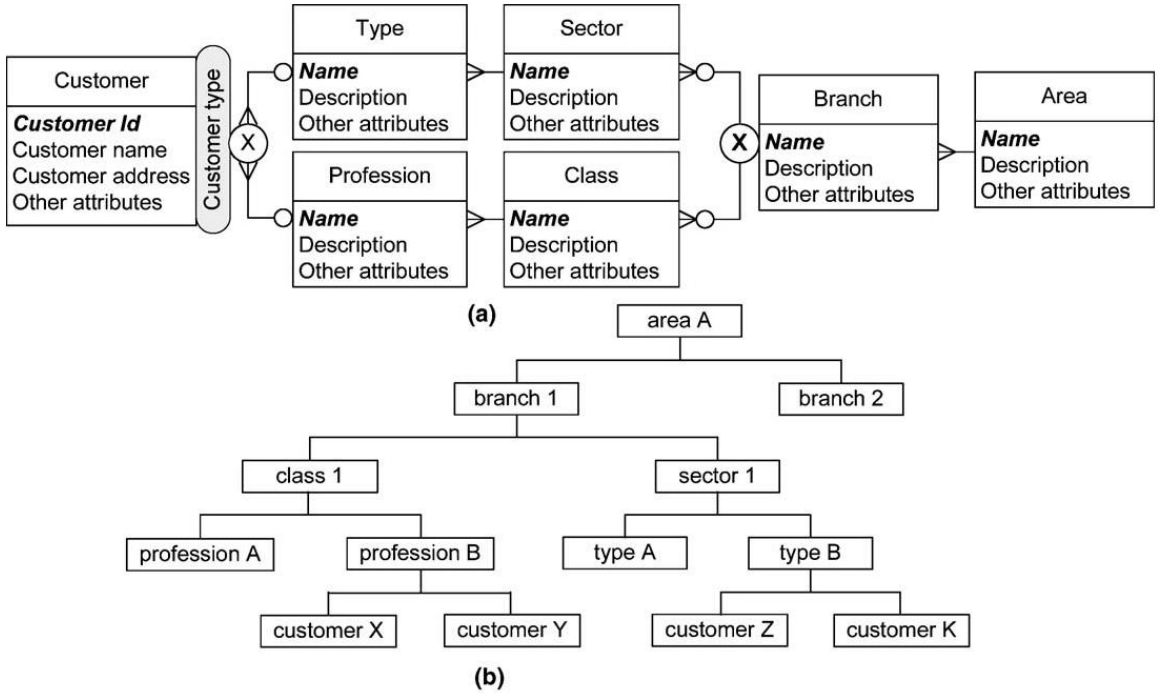

Figure 16: Example for a generalized hierarchy: (a) schema and (b) instances

A special case of generalized hierarchies is a non-covering hierarchy, where at the schema level alternative paths can be achieved by skipping one or more intermediate levels. A child member exactly has one parent member, but the length of the paths from the leaves to the same parent level may vary for different members.

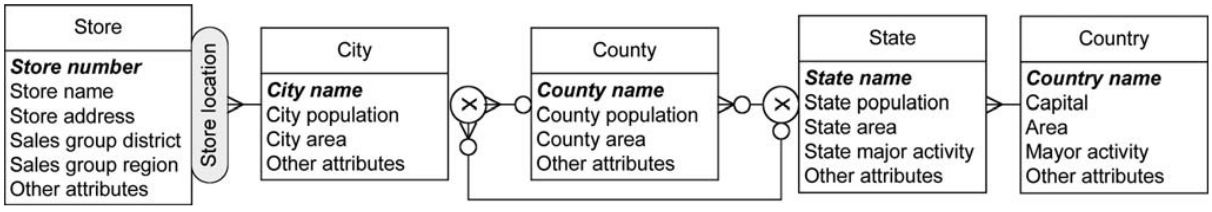

Figure 17: Example for a non-covering hierarchy

In a generalized hierarchy the mapping from a splitting level to a parent level is incomplete, because not all members of the splitting level roll up to a member of each parent level. Thus, completeness is not given so the summarizability condition

is not fulfilled. For this reason, special aggregation mechanisms are required in roll up operations.

2.3.1.1.2 Strictness of simple hierarchies

2.3.1.1.2.1 Strict hierarchies

In a strict hierarchy, there is at the schema level a one-to-many cardinality between all parent and child levels. At the instance level a child member exactly belongs to one parent member and a parent member may have several child members. All previous examples were strict hierarchies, so there is no figure for this hierarchy.

2.3.1.1.2.2 Non-strict hierarchies

If there is at the schema level at least one many-to-many cardinality between a parent and a child level, a non-strict hierarchy is present. Thus a child member can be related to several parent members and also a parent member may have several child members at the instance level, so this hierarchy forms a graph.

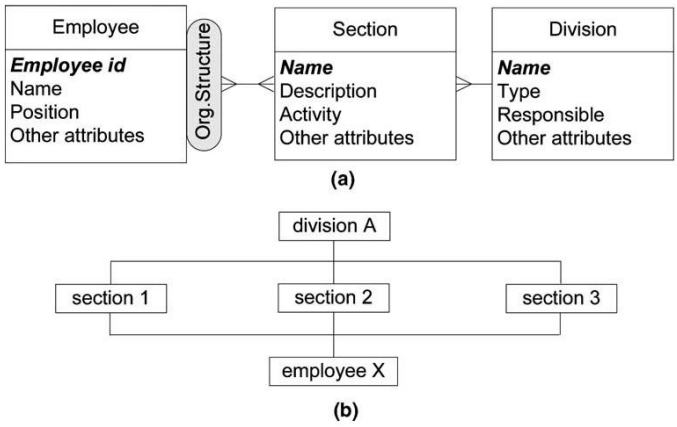

Figure 18: Example for a symmetric non-strict hierarchy: (a) model and (b) instances

For this hierarchy, disjointness is not given, so that the summarizability condition is not fulfilled. Therefore special data structures and algorithms are needed, to aggregate data within this hierarchy. One approach is called bridge table, where the percentage distribution of a child member to several parent members is used to

aggregate the measures. For example employee X in the above figure is assigned to section 1 with 20%, to section 2 with 30% and section 3 with 50%. When calculating the average on the section level for example, this given percentage distribution is considered.

2.3.1.2 Multiple alternative hierarchies

The previously described hierarchies all were specializations of a simple hierarchy. Their common property was that only one path at the schema level is possible. In contrast to them, multiple alternative hierarchies share some levels of several non-exclusive simple hierarchies at the schema level. At the instance level, a child member may belong to more than one parent member and n paths through the hierarchy are possible. As a consequence, a graph is formed. In analysis, the user has to choose one of the alternative paths, because it is semantically not correct to simultaneously traverse the different composing simple hierarchies.

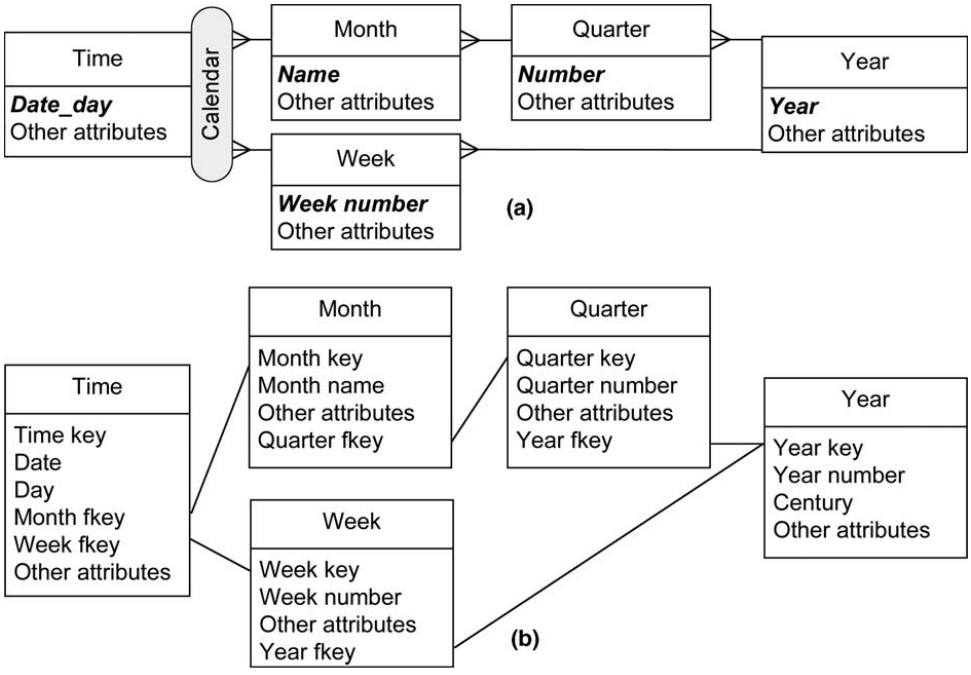

Figure 19: Example for a multiple alternative hierarchy: (a) model and (b) relations

2.3.2 **Parallel hierarchies**

Parallel hierarchies are complementary to individual hierarchies, because not exactly one analysis criterion is used, but rather more analysis criteria can be used within one dimension. These hierarchies can be dependent or independent.

2.3.2.1 Parallel independent hierarchies

In a parallel independent hierarchy, there is at the schema level no connection between the different analysis criteria, which means they do not share levels. So they represent non-overlapping sets of hierarchies, which can be of different kind.

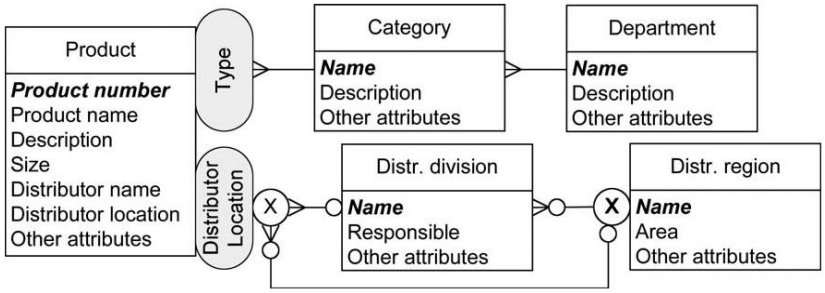

Figure 20: Example for a parallel independent hierarchy

2.3.2.2 Parallel dependent hierarchies

In a parallel dependent hierarchy, there is at the schema level a connection between the different analysis criteria, which means that at the minimum one level of different hierarchies is shared.

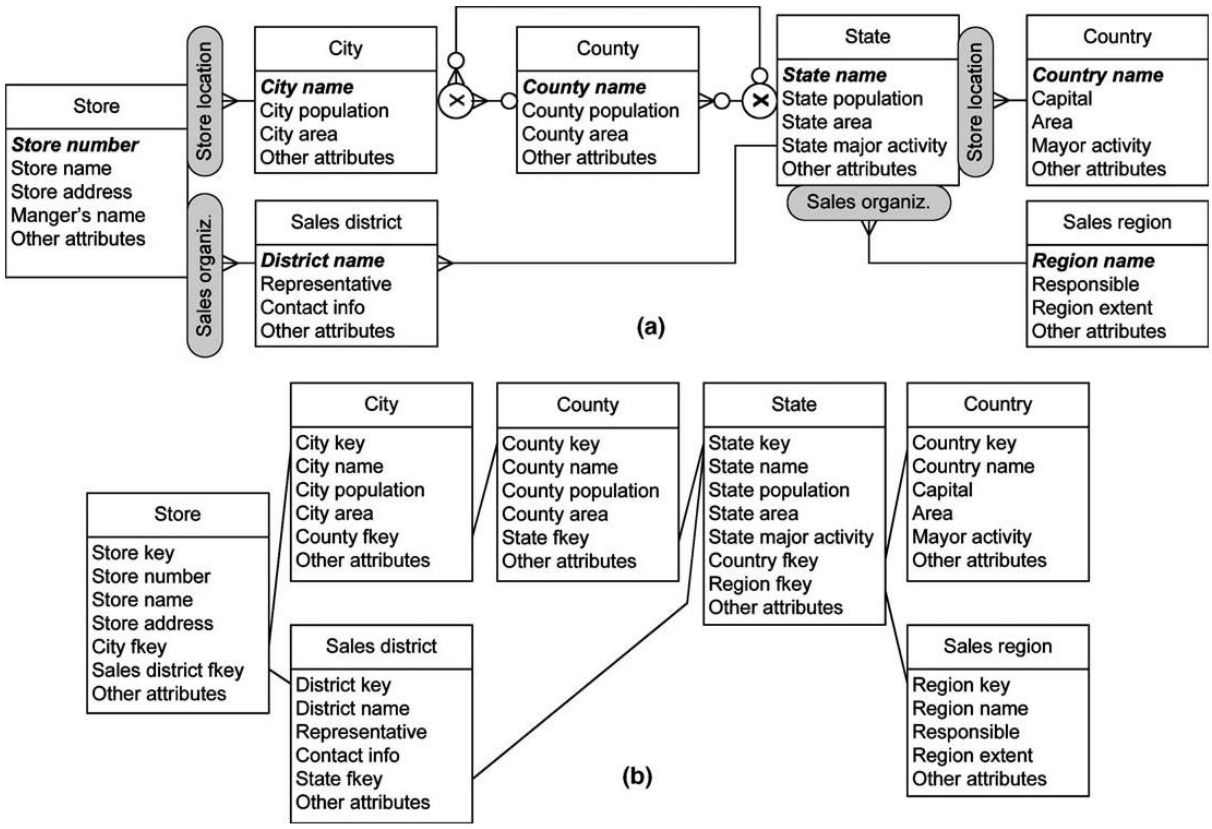

Figure 21: Example for a parallel dependent hierarchy: (a) model and (b) relations

2.3.3 **Usefulness of hierarchies**

After all different types of hierarchies have been introduced; this subsection answers the question how the usefulness of a hierarchy can be measured formally. This has to be explained because the developed approaches generate hierarchies, but make no statement about the usefulness of them. So in this subsection metrics for measuring the usefulness of a hierarchy are introduced and explained formally.

The proposed metrics should be seen as an indicator for a human for the usefulness of a hierarchy. The ultimately usefulness of a hierarchy definitely depends on more aspects:

- User:

  Usefulness always depends on the personal favorites of a user. Somebody preferring for example a clear and excessive structure in a book with many classification levels would possibly prefer rather more than less levels in an OLAP hierarchy.

- Use Case:

  Usefulness also depends on the use case of analyzing the data in an OLAP cube. For example to get an overview and a feeling for the data included in a cube, possibly a balanced hierarchy with some, but not too many levels is reasonable. However, a data analyst searching for specific information on many different levels possibly would prefer a proper hierarchy with many levels.

Because the metrics each time depend on and correspond to different things, it has to be distinguished between the type of metrics and the correspondence of metrics:

- Type of metrics:
  - Quantitative metrics: Can be computed with mathematic formulas, since they fully depend on number and relationships of levels and members of a hierarchy. They are denoted by (n).
  - Qualitative metrics: Have to be estimated by a user, since they depend on the usefulness out of the human view. They are denoted by (q).
- Correspondence of metrics:
  - Metrics corresponding to one approach: They are valid for one approach across all hierarchies that are found with this approach.
  - Metrics corresponding to one found hierarchy: They are valid for one distinct hierarchy and they do not depend on the approach, with which this hierarchy was found.

To compare and apply against each other, the following metrics are normalized on the interval [0;1]. The higher the value is the better the evaluation is. The formulas of the quantitative metrics are explained with an example and the qualitative metrics are explained in prose. It is denoted in brackets, on which element the formula is referring to.

2.3.3.1 Metrics corresponding to one approach

The metrics corresponding to one approach refer to the different approaches to generate hierarchies, described in section 3. The following metrics are used:

- Number of found hierarchies (n):

  $$Number\ of\ found\ hierarchies(dimension) = \frac{h}{h_{max}}$$

h = number of found simple hierarchies in a dimension h_{max} = maximum number of found simple hierarchies in a dimension

The number of found hierarchies of one approach corresponds then to the average of the number of found hierarchies of all dimensions.

- Number of dimensions with at least one hierarchy (n)

  $$\text{Number of dimensions with at least one hierarchy(approach)} = \frac{d_h}{d}$$

d_h = number of dimensions with at least one hierarchy d = total number of dimensions

2.3.3.2 Metrics corresponding to one found hierarchy

Metrics corresponding to one found hierarchy cannot be generalized sine they depend on the type and strictness of a simple hierarchy. Therefore the following table distinguishes strictness on columns and type on rows:

	Strict	**Non-Strict**
Symetric	• Balance (n) • Cardinality (n) • Label-uniqueness (n) • Labels of the analysis criterions (q)	• Label-uniqueness (n) • Labels of the analysis criterions (q)
Asymetric	• Balance of the symmetric part (n) • Cardinalit of the symmetric part (n) • Label-uniqueness (n) • Labels of the analysis criterions (q)	• Label-uniqueness (n) • Labels of the analysis criterions (q)
Generalized	• Label-uniqueness (n) • Labels of the analysis criterions (q)	• Label-uniqueness (n) • Labels of the analysis criterions (q)

Table 4: Metrics of a simple hierarchy depending on type and strictness

To explain the formulas of the metrics, the following abstract example is used. There is one hierarchy with the root, one level and the leafs. Some of the metrics correspond to a certain level, some metrics correspond to a certain member.

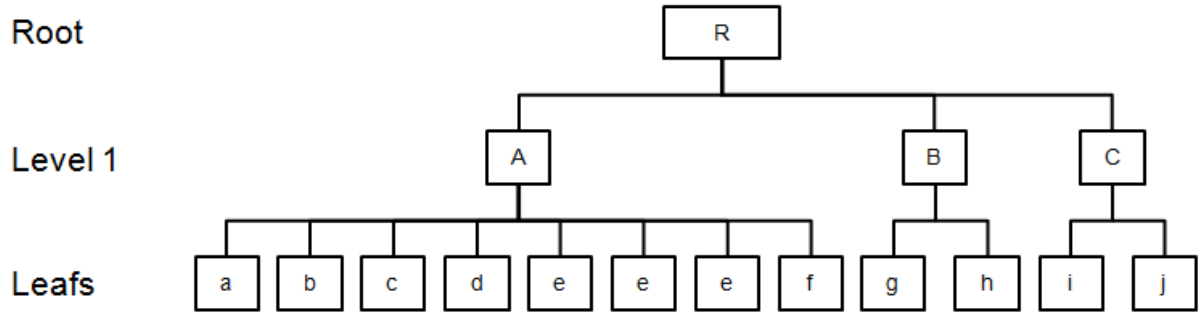

Figure 22: Example of a hierarchy to explain the formulas

- Balance (n):

$$\text{Balance}(\text{level}) = 1 - \frac{\sum_{i=1}^{P} |p_i - \mu|}{(P-1)\mu + (C-\mu)}$$

$P =$ total number of parent members
$p_i =$ number of child members of the i-th member
$\mu =$ average number of child members
$C =$ total number of child members

This metric is changed borrowed from [OrDD06], since it is an indicator for navigation efficiency. Here, the denotation is changed to guarantee more clarity. The balance of level 1 in the example is then calculated as follows:

$$\text{balance}(\text{level } 1) = 1 - \frac{|8-4|+|2-4|+|2-4|}{(3-1)4+(12-4)} = 0{,}5$$

The balance of the whole hierarchy corresponds to the average of the balances of each level except the root and the leaf level.

- Cardinality (n):

$$\text{Cardinality}(\text{member}) = \begin{cases} 0 \text{ if } c \leq 1 \\ \exp^{-\frac{(c-\alpha)^2}{2\beta^2}} \end{cases}$$

$c =$ number of child members

$\alpha =$ parameter for the minimum cardinality
$\beta =$ parameter for the maximum cardinality

This metric is changed borrowed from [OrDD06], since a suitable parent member has a limited but higher than one amount of children. Here, the denotation is changed to guarantee more clarity. The cardinality of member A in the example with the assumption of a minimum cardinality of 2 and a maximum cardinality of 20 children is then calculated as follows:

$$\text{Cardinality}(A) = \exp^{-\frac{(8-2)^2}{2*20^2}} = 0{,}956$$

The cardinality of the whole hierarchy corresponds to the average of the cardinalities of each member except the cardinality for the leaf level.

- Label-uniqueness (n)

$$\text{Label-uniqueness}(\text{member}) = \frac{c_u}{c}$$

$c =$ number of child members
$c_u =$ number of unique-labeled child members

The label-uniqueness of member A in the example is then calculated as follows:

$$\text{label-uniqueness}(A) = \frac{5}{8} = 0{,}625$$

The label-uniqueness of the whole hierarchy corresponds to the average of the label-uniqueness of each member except the label-uniqueness for the leaf level.

3 Expressing OLAP hierarchies in RDF

To express all different types of hierarchies in Linked Data, a proper vocabulary is required. Therefore this section critically validates the recommendation of the RDF Data Cube Vocabulary for expressing hierarchies and proposes an extension for this vocabulary.

3.1 Simple Knowledge Organization System (SKOS)

To express hierarchical structures, the RDF Data Cube Vocabulary recommends the use of Simple Knowledge Organization System (SKOS)[19], a W3C Recommendation for expressing the basic structure and content of concept schemes.

The central elements in SKOS are concepts, which can be defined as ‘units of thougths’, e.g. ideas, meanings, or (categories of) objects and events. Each concept is an instance of the class skos:Concept. Concepts can be grouped together to a skos:ConceptScheme to generate a predefined vocabulary, such as thesauri or classification schemes.

To represent hierarchical relationships between concepts, the properties skos:narrower and skos:broader are used. The subject of the property skos:narrower is the more generalized concept, so this property should be read as ‘has narrower concept’. The two properties skos:narrower and skos:broader are in each case inverse to each other and each concept may have both, many narrower and many broader concepts.

These two properties are not defined to be transitive. So if a concept A is broader than a concept B, which is itself broader than a concept C, the concept A is not broader than the concept C. To express these semantics, it is required to use the two properties skos:narrowerTransitive and skos:broaderTransitive.

Each member of a dimension in the RDF Data Cube Vocabulary can be defined as skos:Concept. So the actual hierarchy is not expressed with the RDF Data Cube Vocabulary but rather with SKOS.

[19] http://www.w3.org/TR/skos-primer/

It must be noticed that all concepts that belong to a concept scheme are potential the members of the dimension. A member of a dimension is in union a concept in SKOS. If a dimension is linked to a concept scheme and there is a member within this dimension that is not included in this concept scheme, this is a mistake in modeling.

In SKOS it is not forbidden to model loops of broader and narrower relationships. E.g. a concept A is broader than a concept B that is itself broader than the concept A. In OLAP hierarchies, where the aim is to roll up values, this would not be meaningful and cannot be expressed. So if loops in a concept scheme are found, there does not exist a standard way to handle this problematic and a human user has to be interacted, who defines the relationships in the hierarchy.

Furthermore, there may exist some concepts with broader and narrower relationships that are not members of the dimension. If they are not relevant for the remaining hierarchy with the existing members, they may be deleted. There is also the possibility to retain them for completeness reasons. So if some additional members, for which the concepts have been deleted, occur in a future transformation, the hierarchy has not to be constructed anew. This has to be decided beforehand.

3.2 Weaknesses of SKOS

Although SKOS is the recommended way of representing hierarchical structures in statistical data of the RDF Data Cube Vocabulary, it has the following disadvantages when modeling hierarchies without further metadata:

- No membership of a concept to a certain level:
  SKOS works on instance level. This means that a certain concept does not belong to a certain level. As a consequence, the following hierarchies can't be expressed with SKOS because there is no assignment of concepts to levels:
    - Parallel hierarchies
    - Multiple alternative hierarchies
    - Generalized hierarchies

  With reference to Figure 16 of a generalized hierarchy above, for example class 1 and sector 1 have the same broader concept branch 1. When generating the hierarchy with SKOS, it is not clear that the concepts for class

1 and sector 1 do not belong to the same level. Class 1 and sector 1 would be narrower concepts from branch 1, but cannot be distinguished by the levels. As a further consequence, also labels for levels are missing, since a concept in SKOS is not assigned to a certain level and there is no information about levels.

- No Attributes of a level:
  In the multidimensional model of [MaZi05], each level can have besides its key attribute certain descriptive attributes and characteristics for these attributes for each instance of this level. SKOS however, does not make a statement to descriptive attributes of a certain concept. This means that using this approach, which is a W3C recommendation, descriptive attributes cannot be defined and their characteristics cannot be filled when generating the levels of a hierarchy with this approach.
  With reference to Figure 16 of a generalized hierarchy above, e.g. the customer address is an attribute of the customer ID in the level customer. In SKOS, it cannot be expressed that the customer address is a descriptive attribute of the customer ID and its characteristic (e.g. ‘Town Street 22’) cannot be linked to a certain customer.
  Furthermore, to express generalized hierarchies, it is required that levels can be distinguished by its descriptive attributes. That being not the case, generalized hierarchies and its special cases non-covering hierarchies cannot be expressed with SKOS.
- Member of a dimension is a concept:
  A member of a dimension in the RDF Data Cube Vocabulary is simultaneous a concept in SKOS. This means that all relationships of the concepts expressed within SKOS have also significance for the members in the RDF Data Cube Vocabulary. This must not always be semantically correct.
  For example there is a concept for a city that has a state as broader concept. On the semantic level, the city itself does not have a broader concept but it is the concept for the city that has a broader concept.
- Relationships between concepts independent of concept schemes:
  The hierarchical relationships between two concepts are universal valid, independent of concept schemes. For example there should be defined the

two concept schemes ex:StrictScheme for a strict hierarchy country→language and ex:NonStrictScheme for a non-strict hierarchy country→language, but the last relationship in the following example should only be valid for the concept scheme ex:NonStrictScheme. This is not expressible with SKOS, since the relationships are universal valid.

ex:Germany	skos:inScheme	ex:StrictScheme.
ex:Germany	skos:inScheme	ex:NonStrictScheme.
ex:Switzerland	skos:inScheme	ex:StrictScheme.
ex:Switzerland	skos:inScheme	ex:NonStrictScheme.
ex:Germany	skos:broader	ex:German.
ex:Switzerland	skos:broader	ex:Swiss.
ex:Switzerland	skos:broader	ex:German.

Summarizing the above disadvantages of SKOS, the following figure gives an overview of such hierarchies that can be expressed in SKOS because of the attribute and level problematic:

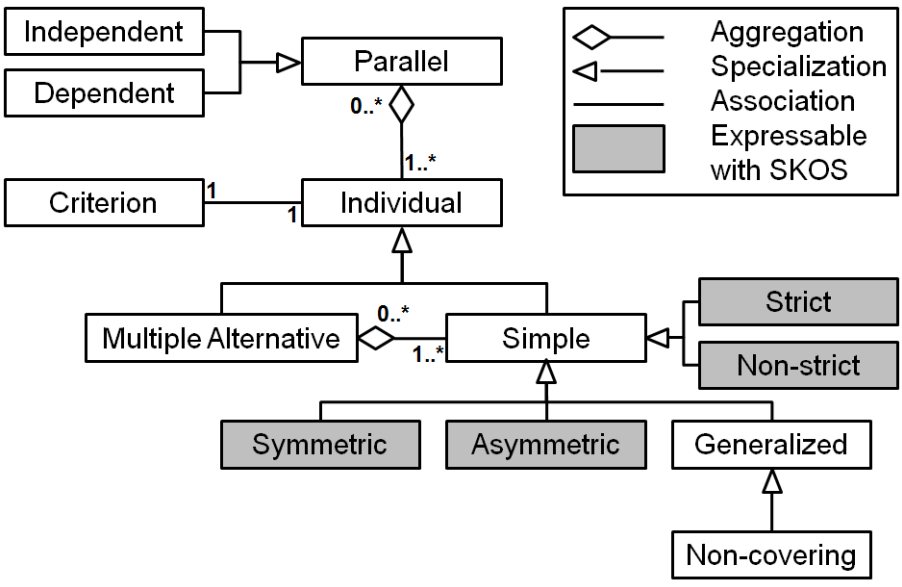

Figure 23: Hierarchies expressible with SKOS

3.3 Proposed extensions to SKOS

To express all hierarchies with SKOS and enrich the way, the RDF Data Cube vocabulary uses SKOS, the following elements are needed to improve it. Besides the use of SKOSE, which is the proposed extension of this master thesis, also SKOSCLASS[20], which is already a proposed extension to SKOS, is reused in this master thesis:

- Introducing levels:

  To assign concepts to a level, information about levels is required. SKOSCLASS suggests the class skosclass:ClassificationLevel, whose instances are representing the levels of the hierarchy. Since a concept has no information that indicates the membership to a level, the property skos:member is suggested to assign a concept to a certain level. Levels are then assigned via the property skos:inScheme to a concept scheme and via skosclass:depth to a numeric value, representing the root distance. Furthermore, a label could be assigned to a certain level via rdfs:label. There are two ways of finding labels for a level:

  - Using generic and continuous keys:

    One way to find the labels for the hierarchy levels is using generic and continuous keys. For example there is a concept scheme with the following relationships for concepts:

ex:A	skos:broader	ex:C.
ex:B	skos:broader	ex:C.

    The concepts ex:A and ex:B could belong to the same level. Since there is nothing available in SKOS that indicates the level, a useful label could not be determined. So generic and continuous keys have to be used, e.g. ‘Level 1’ for the level to which the concept ex:C belongs to and ‘Level 2’ for the level to which the concepts ex:A and ex:B belong to.

  - Using labels of the concepts:

[20] http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS

Another way is to determine the label for the level with the help of the labels of the concepts. If all concepts, which belong to a certain level, have similar labels, the label for the level could be determined using these labels. E.g. there is a SKOS concept scheme with the following concepts and relationships:

ex:Branch1	skos:broader	ex:Bank1.
ex:Branch2	skos:broader	ex:Bank1.

The label for the level, to which the two concepts ex:Branch1 and ex:Branch2 belong to, could be ‘Branch’.

- Introducing descriptive attributes:
  To define descriptive attributes of a certain level and fill them with characteristics, it is required that SKOS should have the possibility to define additional properties and fill them with characteristics. There should exist a class for descriptive attributes. A suggestion for this class is skose:Attribute. All instances of this class are properties. With such a property a concept could be linked to an object that indicates the characteristic of this attribute. The property of the triple is the attribute and the object of the triple is the characteristic of the attribute. Each instance of the class skosclass:ClassificationLevel should be linked via the property skose:hasAttribute to an instance of skose:Attribute.
  With reference to Figure 16 of a generalized hierarchy above, e.g. the customer address is a descriptive attribute of the customer ID in the level customer and the characteristic for one customer 123456 is ‘Town Street 22’. This could be expressed as follows:

ex:customer	rdf:type	skosclass:ClassificationLevel;
	skose:hasAttribute	ex:customerAdress.
ex:customerAdress	rdf:type	skose:Attribute.
ex:123456	rdf:type	skos:Concept;
	skos:member	ex:customer;
	ex:customerAdress	“Town Street 22”.

- Separation of members in the RDF Data Cube Vocabulary and concepts in SKOS:

  For the reasons above, the members of a dimension in the RDF Data Cube Vocabulary should be separated from the concepts in SKOS. Therefore the URIs for the members should distinguish from the URIs for the concepts. Furthermore, for each occurring value of a literal, a separate URI is required. Also a property is required that assigns the SKOS concepts to members of a dimension. For reuse reasons, this master thesis suggests to use already standardized properties for this assignment:

  - rdfs:seeAlso:

    If the dimension member is an URI, the property rdfs:seeAlso is used. The property skos:exactMatch or another skos:mappingRelation is not used, because domain and range of these properties is each time a skos:Concept and the actual dimension member should not be defined as skos:Concept.
  - rdfs:label:

    If the dimension member is a literal, the property rdfs:label is used. The property skos:notation is not used, because by convention it is only used for user-defined data types and this master thesis supports all data types, e.g. xsd:date.

  Hence, also several concepts of several schemes in SKOS could be assigned to a member without using owl:sameAs.
- Reification of relationships between concepts:

  AS described above, relationships between concepts are independent of concept schemes. To express several concept schemes with the same concepts, but different relationships between concepts, relationships are reificated using skosclass:hasSource/TargetConcept and are defined as skosclass:ConceptAssociation, which can be assigned to the relevant concept scheme. For example the relationships of the above example are expressed as follows, whereby the last relationship is only assigned to the concept scheme ex:NonStrictScheme.

ex:Germany_German	rdf:type	skosclass:ConceptAssociation;
	skosclass:hasSourceConcept	ex:Germany;
	rdf:predicate	skos:broader;
	skosclass:hasTargetConcept	ex:German;
	skos:inScheme	ex:StrictScheme;
	skos:inScheme	ex:NonStrictScheme.
ex:Switzerland_Swiss	rdf:type	skosclass:ConceptAssociation;
	skosclass:hasSourceConcept	ex: Switzerland;
	rdf:predicate	skos:broader;
	skosclass:hasTargetConcept	ex:Swiss;
	skos:inScheme	ex:StrictScheme;
	skos:inScheme	ex:NonStrictScheme.
ex:Switzerland_German	rdf:type	skosclass:ConceptAssociation;
	skosclass:hasSourceConcept	ex: Switzerland;
	rdf:predicate	skos:broader;
	skosclass:hasTargetConcept	ex:German;
	skos:inScheme	ex:NonStrictScheme.

3.4 Resulting vocabulary

To represent all proper OLAP hierarchies with RDF, the vocabularies of SKOS and SKOSCLASS are used and further extended by SKOSE. Since SKOS is already a W3C Recommendation, SKOSCLASS is as yet a proposed extension to SKOS. SKOSE is a further extension, proposed by this master thesis. The combination of these three vocabularies is used for precondition B, to express found hierarchies with RDF. The following table gives an overview of the used classes and properties.

As it is common practice in many data warehouses, an element (e.g. member, level, hierarchy) may consist of a (technical) key, a (human-readable) label and a (human-readable) description. In each time, the URIs of these elements are used as keys, rdfs:label as label and rdfs:comment as description.

Element or function	Class or property	Name
hierarchy	skos:ConceptScheme	cs/
assigning a hierarchy to a dimension	qb:codeList	-
concepts for members	skos:Concept	co/
linking a concept to its member in case of a resource	rdfs:seeAlso	-
linking a concept to its member in case of a literal	rdfs:label	-
assigning a concept to a hierarchy	skos:inScheme	-
level	skosclass:ClassificationLevel	cl/
assigning a concept to a level	skos:member	-
assigning a level to a hierarchy	skos:inScheme	-
indicating the root distance of a level	skosclass:depth	-
relationship between two concepts	skosclass:ConceptAssociation	ca/
children/parent of a relationship	skosclass:hasSourceConcept	-
hierarchical type of a relationship	skos:broader/skos:narrower	-
parent/children of a relationship	skosclass:hasTargetConcept	-
assigning a relationship to a hierarchy	skos:inScheme	-
descriptive attribute	skose:Attribute	da/
assigning an attribute to a level	skose:hasAttribute	-

Table 5: Classes and properties to express all proper OLAP hierarchies in RDF

The central element is a skos:ConceptScheme, which represents an OLAP hierarchy. To assign a hierarchy to a dimension of a cube, the property qb:codeList is used. As described above in the formal definition, an OLAP hierarchy consists of members, levels, assignments of members to levels and relationships between members. Therefore the following elements are used to represent all different types of hierarchies with RDF.

Members of a dimension are represented as concepts. To avoid that a member of a dimension in the RDF Data Cube Vocabulary is simultaneous a concept in SKOS, concepts are linked to their corresponding members of a dimension via rdfs:seeAlso in case of the member being a resource or via rdfs:label in case of the member being a literal. A concept is linked via skos:inScheme to a certain hierarchy.

Levels are represented by skosclass:ClassificationLevel. Concepts are assigned to a certain level via skos:member and levels are assigned to a certain hierarchy via skos:inScheme. This means that concepts are assigned also indirectly (by way of levels) to a hierarchy. The property skosclass:depth indicates the root distance of a particular level, meaning accordingly to the formal definition the number of levels without skipping an intermediate level between the root and a particular level. The uppermost level has root distance one. Because the depth of a level in a multiple alternative or parallel hierarchy is dependent of the path from this level to the root, skosclass:depth is only set for levels of a simple hierarchy, where the depth is unambigous.

Because relationships between concepts depend on a specific hierarchy and are not generally valid, they have to be reificated. A skosclass:ConceptAssociation represents a relationships between two concepts and links via skosclass:hasSourceConcept and skosclass:hasTargetConcept to the children or parent of a relationship. To determine whether the source/target concept is the child/parent of the relationship, rdf:subject links a relationship to skos:broader/skos:narrower, indicating the hierarchical type of the relationship. To assign a certain relationship to a hierarchy, the property skos:inScheme is used.

Descriptive attributes are represented by skose:Attribute. To assign descriptive attributes to a level, the property skose:hasAttribute is used. All descriptive attributes are properties, which could link concepts to their characteristics of these attributes.

To guarantee a standardized naming of the different resources, the created URIs of the instances of classes are concatenations of the following naming convention:

- ‘http://hierarchie.org/’
- [Name of Table 5 ‘Classes and properties to express all proper OLAP hierarchies’ above, indicating the class of the resource, e.g. ‘co/’ for a concept]
- [Free Part: Relevant substring(s) of related URI(s) without ‘http://’ and replacement of dots and hashes by slashes or useful free names]

For example, the concept for the resource <http://dbpedia.org/resource/Germany> would be <http://hierarchie.org/co/dbpedia/org/resource/Germany>.

4 Transforming Linked Data into OLAP hierarchies

The transformation of the explicitly expressed hierarchies with the extended SKOS vocabulary into the OLAP4J[21] standard is described in the following subsections in a way that they can be used in OLAP systems.

4.1 OLAP4J

OLAP4J is an open Java application programming interface (API) for accessing multi-dimensional data. It intends to standardize the access on multidimensional data on any OLAP server in a way that an OLAP application written in Java for one server can easily be switched to another.

Generally speaking, an OLAP application interacts with an OLAP server by means of MDX statements. OLAP4J provides the possibility to create a query by parsing an MDX statement or to build a query by manipulating an MDX parse tree, whereby an MDX parser library allows an easy conversion of an MDX string to and from a parse tree.

XML for Analysis (XMLA)[22] is also a standardized API, designed specifically for the data access interaction between a client application and a multi-dimensional data provider over the web and had gained a broad support of companies like Hyperion, Microsoft, SAP and SAS.

Both, XMLA and OLAP4J allow an application to execute OLAP queries and to browse the metadata of an OLAP schema. Since XMLA is a low-level web-service API which leaves a lot of work to the application writer, OLAP4J provides advanced functions for parsing MDX, building and transforming MDX query models and for mapping result sets into graphical layouts such as pivot tables. However, OLAP4J can easily be added to an XMLA back-end.

In this master thesis OLAP4J serves as interface between the explicitly expressed hierarchies with RDF on the one side and an OLAP application on the other side. Corresponding to the API, the schema result sets are filled with the relevant

[21] http://www.olap4j.org/
[22] http://news.xmlforanalysis.com/what-is-xmla

information. Concerning hierarchies, the following three methods are the most important ones:

- getHierarchies
- getLevels
- getMembers

Corresponding to one implementation[23], OLAP4J supports the following four hierarchy types:

- Balanced:
  In a balanced hierarchy, the parent of each member comes from the level immediately above the member.
- Ragged:
  In a ragged hierarchy, the parent of a member can come from any level above the level of the member, not just from the level immediately above. This type of hierarchy can also be referred to as ragged-balanced, because levels still exist.
- Unbalanced:
  In an unbalanced hierarchy, the concept of levels is not applied.
- Network:
  The distinguishing feature of a network hierarchy is that nodes can contain more than one parent.

4.2 Mapping between SKOS and OLAP4J

As described above, the standards XMLA and OLAP4J support four different types of hierarchies. However, the hierarchies in a multidimensional model which were conceptually introduced in section 2.3 can be more complex. For this reason, a mapping between the conceptual model and OLAP4J is required. The found hierarchies, which were expressed with SKOS and the proposed extensions, are interpreted analogously the conceptual model.

[23] http://msdn.microsoft.com/en-us/library/ms725445%28v=VS.85%29.aspx

The following table gives an overview of the mapping between the hierarchies in the extended SKOS vocabulary respectively the conceptual model and OLAP4J. It has to be noticed that the naming and the understanding of the hierarchies is different, but OLAP4J basically supports all simple hierarchies of the extended SKOS vocabulary respectively the conceptual model:

SKOS/Conceptual Model			OLAP4J	Differences
parallel independent			-	A parallel independent hierarchy can be transformed in several simple hierarchies, which can then be used in OLAP4J.
parallel dependent			-	A parallel dependent hierarchy can be split in several simple hierarchies, which can then be used in OLAP4J.
multiple alternative			-	A multiple alternative hierarchy can be split in several simple hierarchies, which can then be used in OLAP4J.
simple	strict	symmetric	balanced	-
		asymmetric	unbalanced	-
		generalized (except non-covering)	balanced or unbalanced	All levels on the same stage of a generalized hierarchy have to be transformed in one common level in the balanced or unbalanced hierarchy.
		non-covering	ragged	All levels on the same stage of a non-covering hierarchy have to be transformed in one common level in the ragged hierarchy.
	non-strict	symmetric asymmetric generalized (inclusive non-covering)	network	All levels on the same stage of a generalized hierarchy have to be transformed in one common level in the network hierarchy.

Table 6: Hierarchy mapping between SKOS and OLAP4J

The metamodel of hierarchy classification [Figure 12] shows that both, multiple alternative and parallel hierarchies are aggregations of simple hierarchies. Since there is no pendant for these complex hierarchies in OLAP4J, this relationship is used to reduce the more complex hierarchies into simple hierarchies. After reducing multiple alternative and parallel hierarchies to simple hierarchies, all levels of a generalized hierarchy on the same stage have to be integrated in one common level and the depths of levels have to be set. The following subsections specify the transformation of the hierarchies in a way that they can be used in OLAP4J.

4.2.1 Transforming parallel hierarchies in individual hierarchies

Parallel hierarchies are split in several individual hierarchies, whereby each analysis criterion results in an own individual hierarchy. All resulting individual hierarchies that are multiple alternative hierarchies must then be split further into several simple hierarchies.

- Parallel independent hierarchies:

  Since a parallel independent hierarchy does not share levels, all levels that are in the paths of one analysis criterion are part in the resulting individual hierarchy. For example the above described parallel independent hierarchy [Figure 20] results in the following two simple hierarchies:

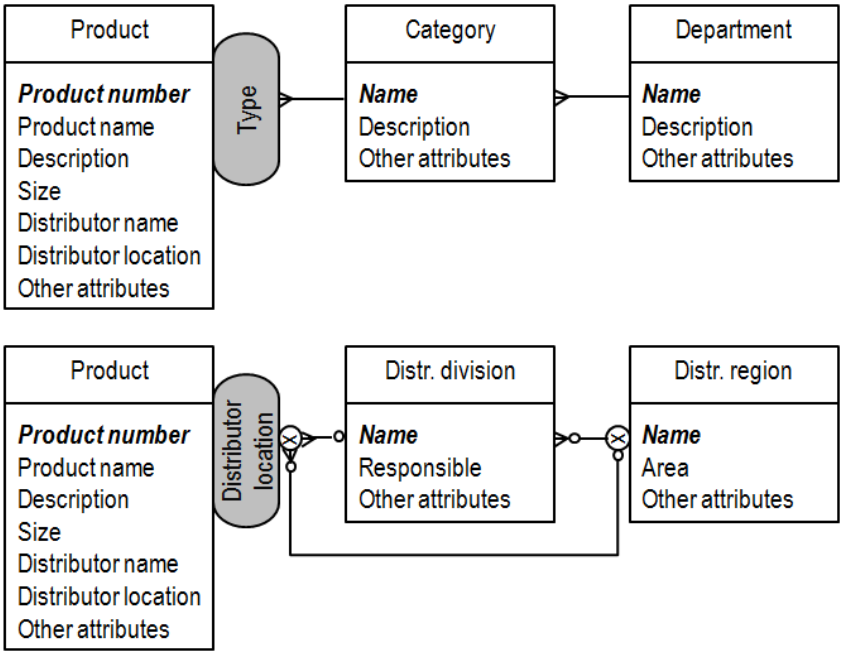

Figure 24: Example for a transformation of a parallel independent hierarchy

- Parallel dependent hierarchies:

  Since a parallel dependent hierarchy shares levels, all these levels are part of all resulting individual hierarchies. For example the level state in the above described parallel dependent hierarchy [Figure 21] is part in both resulting simple hierarchies:

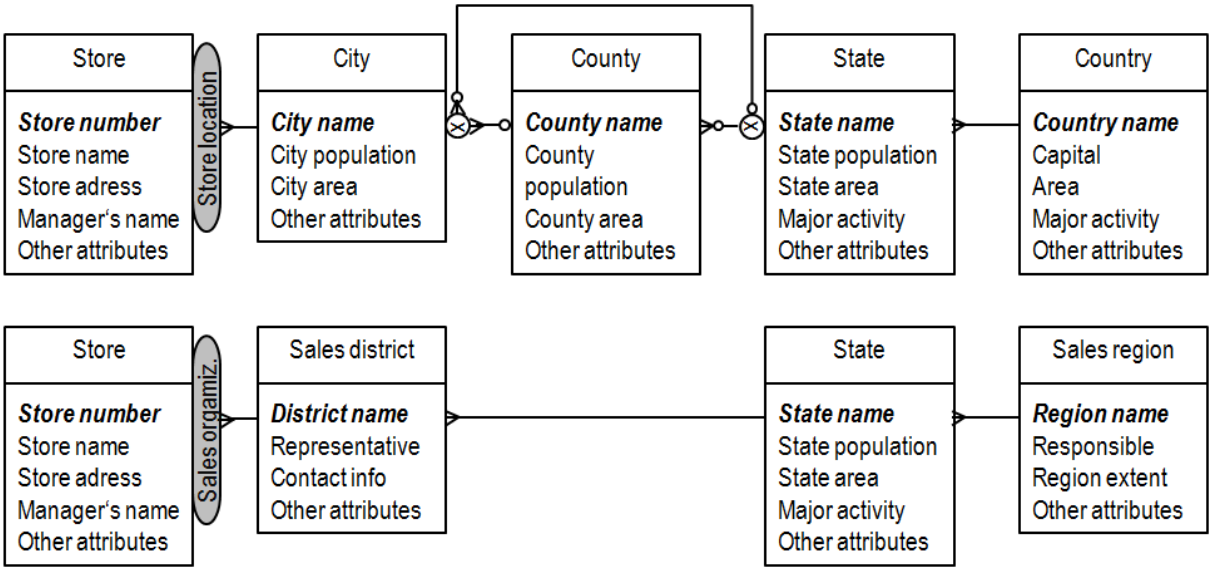

Figure 25: Example for a transformation of a parallel dependent hierarchy

4.2.2 Transforming multiple alternative hierarchies in simple hierarchies

Since it is semantically not correct in such hierarchies to simultaneously traverse the different composing hierarchies, the user has to choose one of the alternative hierarchies. Therefore several simple hierarchies are generated, whereby the shared levels are part of each simple hierarchy. For example, the above described multiple alternative hierarchy [Figure 19] for the time dimension results in the following two simple hierarchies:

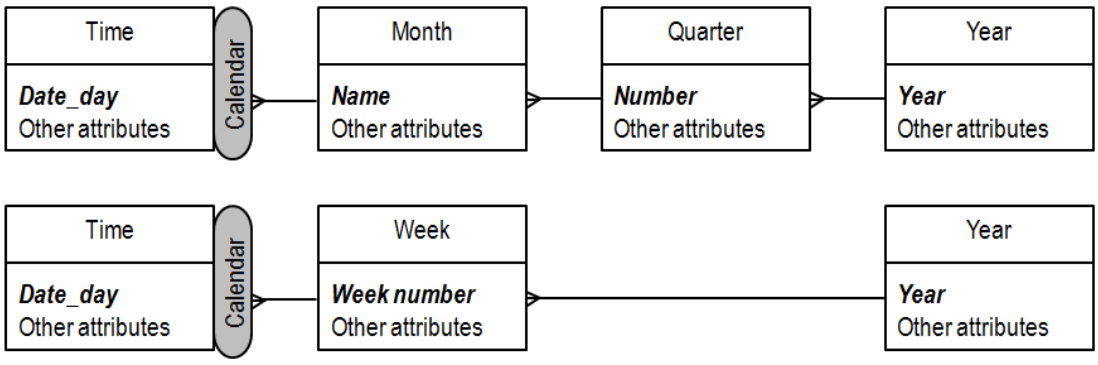

Figure 26: Example for a transformation of a multiple alternative hierarchy

4.2.3 Transforming generalized hierarchies

Since OLAP4J does not allow different levels with the same depth, all levels on the same stage have to be integrated in one common level. For example, the above

described generalized hierarchy [Figure 16] results in the following symmetric hierarchy:

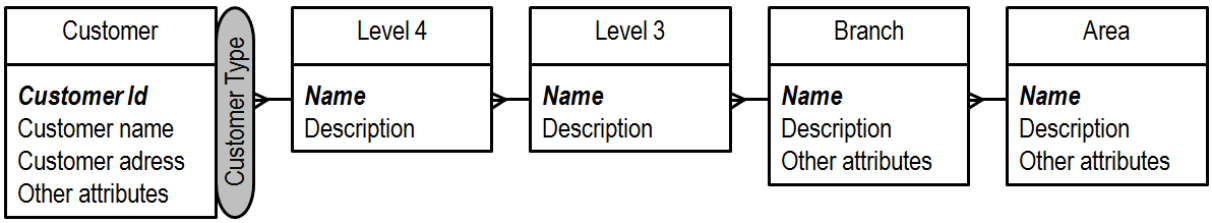

Figure 27: Example for a transformation of a generalized hierarchy

The two levels ‘Type’ and ‘Profession’ on stage 4 result in one common level ‘Level 4’ and the two levels ‘Sector’ and ‘Class’ result in one common level ‘Level 3’. All attributes that have been part of both levels (e.g. ‘Description’) result in an attribute in the common level. Other attributes, which have been level-specific, are not part of the new common level.

4.3 OLAP4J methods

To use the constructed hierarchies the following subsections specify the OLAP4J methods:

- getHierarchies
- getLevels
- getMembers

To use the specified SPARQL queries for the OLAP4J methods concerning hierarchies, the following preparations have to be done:

- Transformation of hierarchies of the conceptual model into OLAP4J hierarchies:
  As described above, the hierarchies of the conceptual model have to be transformed into OLAP4J conform hierarchies corresponding to the specified mapping. All more complex hierarchies are reduced to simple hierarchies and all levels on the same stage in a generalized hierarchy have to be integrated in one common level. To select all hierarchies, which fulfill this assumption,

the following SPARQL query can be used. Only the selected hierarchies of this query are relevant hierarchies for the use in OLAP4J. The results of the queries for the methods can be filtered about the result of this query. In this query, it is grouped by ?HIERARCHY_UNIQUE_NAME and ?LEVEL_NUMBER to count all levels per stage per hierarchy. The having condition is then used to select only the required hierarchies (?HIERARCHY_UNIQUE_NAME).

```
Select distinct
#Relevant hierarchies for OLAP4J
?HIERARCHY_UNIQUE_NAME
where{
Select distinct
?HIERARCHY_UNIQUE_NAME
?LEVEL_NUMBER
(count(?LEVEL_NUMBER) AS ?numberOfLevelsOnTheSameStage)

where{
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
?LEVEL_UNIQUE_NAME rdf:type skosclass:ClassificationLevel.
?LEVEL_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?LEVEL_UNIQUE_NAME skosclass:depth ?LEVEL_NUMBER.
}
GROUP BY
?HIERARCHY_UNIQUE_NAME
?LEVEL_NUMBER
HAVING(?numberOfLevelsOnTheSameStage = 1)}
```

- Only skos:broader instead of skos:narrower properties are used:
  Since there is the possibility to express hierarchical structures bottom-up or top-down, the used properties have to be harmonized to have the possibility to use only skos:broader instead of skos:narrower. This can be done with the following SPARQL construct query, whereby the resulting triples have to be added in the triple store.

```
Construct {
#Transforming skos:narrower in skos:broader concept associations
?relationshipBroader rdf:type skosclass:ConceptAssociation.
?relationshipBroader skos:inScheme ?conceptScheme.
?relationshipBroader skosclass:hasSourceConcept ?member.
```

```
?relationshipBroader rdf:predicate skos:broader.
?relationshipBroader skosclass:hasTargetConcept ?parent.
}
where{
?relationship rdf:type skosclass:ConceptAssociation.
?relationship skos:inScheme ?conceptScheme.
?relationship skosclass:hasSourceConcept ?parent.
?relationship rdf:predicate skos:narrower.
?relationship skosclass:hasTargetConcept ?member.
BIND(URI(CONCAT(STR(?relationship),"_BROADER")) AS ?relationshipBroader)
}
```

4.3.1 **getHierarchies**

Since it is not trivial to determine the structure of the resulting hierarchy for the use in OLAP4J, this part is separated from determining the remaining variables. For each of the four different structures, an own SPARQL query is created to select the hierarchies, having the particular structure:

- MD_STRUCTURE_FULLYBALANCED (0):
  In this query it has to be proofed if each member (?MEMBER_UNIQUE_NAME) has a child (possibly indirect) on the leaf level. Therefore, a property path is used and the results have to be grouped by ?HIERARCHY_UNIQUE_NAME and ?STRUCTURE. To select only the required hierarchies, the having condition is used.

```
Select distinct
#Structure of concept scheme  = MD_STRUCTURE_ FULLYBALANCED (0)
?HIERARCHY_UNIQUE_NAME ?STRUCTURE
where{
Select distinct
?HIERARCHY_UNIQUE_NAME ?MEMBER_UNIQUE_NAME
(max(xsd:integer(?CHILD_LEVEL_NUMBER))AS ?LEAF_LEVEL_NUMBER) ?STRUCTURE
where{
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:member ?MEMBER_LEVEL.
?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_LEVEL rdf:type skosclass:ClassificationLevel.
#?MEMBER_LEVEL skosclass:depth "1".
?MEMBER_UNIQUE_NAME  (^skosclass:hasTargetConcept/skosclass:hasSourceConcept)*
?CHILD_UNIQUE_NAME.
```

```
?CHILD_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?CHILD_UNIQUE_NAME skos:member ?CHILD_LEVEL.
?CHILD_UNIQUE_NAME rdf:type skos:Concept.
?CHILD_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?CHILD_LEVEL rdf:type skosclass:ClassificationLevel.
?CHILD_LEVEL skosclass:depth ?CHILD_LEVEL_NUMBER.
BIND("0" AS ?STRUCTURE)
} GROUP BY ?HIERARCHY_UNIQUE_NAME ?MEMBER_UNIQUE_NAME ?STRUCTURE
} GROUP BY ?HIERARCHY_UNIQUE_NAME ?STRUCTURE
HAVING (
(max(xsd:integer(?LEAF_LEVEL_NUMBER)))*(count(xsd:integer(?LEAF_LEVEL_NUMBER)
)) = (sum(xsd:integer(?LEAF_LEVEL_NUMBER)))
)
```

- MD_STRUCTURE_RAGGEDBALANCED (1):

  In this query it has to be proofed if there is a difference between the depths of levels of a concept association which is higher than one.

```
Select distinct
#Structure of concept scheme  = MD_STRUCTURE_RAGGEDBALANCED (1)
?HIERARCHY_UNIQUE_NAME ?STRUCTURE
where{
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
?relationship rdf:type skosclass:ConceptAssociation.
?relationship skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?relationship skosclass:hasSourceConcept ?MEMBER_UNIQUE_NAME.
?relationship rdf:predicate skos:broader.
?relationship skosclass:hasTargetConcept ?PARENT_UNIQUE_NAME.

?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:member ?MEMBER_LEVEL.
?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_LEVEL rdf:type skosclass:ClassificationLevel.
?MEMBER_LEVEL skosclass:depth ?MEMBER_LEVEL_NUMBER.

?PARENT_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?PARENT_UNIQUE_NAME skos:member ?PARENT_LEVEL.
?PARENT_UNIQUE_NAME rdf:type skos:Concept.
?PARENT_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?PARENT_LEVEL rdf:type skosclass:ClassificationLevel.
?PARENT_LEVEL skosclass:depth ?PARENT_LEVEL_NUMBER.

BIND("1" AS ?STRUCTURE)
FILTER(xsd:integer(?MEMBER_LEVEL_NUMBER)                                -
(xsd:integer(?PARENT_LEVEL_NUMBER)) >1)
}
```

- MD_STRUCTURE_UNBALANCED (2):

  In this query it has to be proofed if there is a member on a higher level(?MEMBER_UNIQUE_NAME) which has no child (possibly indirect) on the leaf level. This query is nearly the same to the first one, with the difference of the reverse having condition.

```
Select distinct
#Structure of concept scheme  = MD_STRUCTURE_UNBALANCED (2)
?HIERARCHY_UNIQUE_NAME ?STRUCTURE
where{
Select distinct
?HIERARCHY_UNIQUE_NAME ?MEMBER_UNIQUE_NAME
(max(xsd:integer(?CHILD_LEVEL_NUMBER))AS ?LEAF_LEVEL_NUMBER) ?STRUCTURE
where{
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:member ?MEMBER_LEVEL.
?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_LEVEL rdf:type skosclass:ClassificationLevel.
#?MEMBER_LEVEL skosclass:depth "1".
?MEMBER_UNIQUE_NAME  (^skosclass:hasTargetConcept/skosclass:hasSourceConcept)*
?CHILD_UNIQUE_NAME.

?CHILD_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?CHILD_UNIQUE_NAME skos:member ?CHILD_LEVEL.
?CHILD_UNIQUE_NAME rdf:type skos:Concept.
?CHILD_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?CHILD_LEVEL rdf:type skosclass:ClassificationLevel.
?CHILD_LEVEL skosclass:depth ?CHILD_LEVEL_NUMBER.

BIND("2" AS ?STRUCTURE)
} GROUP BY ?HIERARCHY_UNIQUE_NAME ?MEMBER_UNIQUE_NAME ?STRUCTURE
} GROUP BY ?HIERARCHY_UNIQUE_NAME ?STRUCTURE
HAVING (
(max(xsd:integer(?LEAF_LEVEL_NUMBER)))*(count(xsd:integer(?LEAF_LEVEL_NUMBER)
)) != (sum(xsd:integer(?LEAF_LEVEL_NUMBER)))
)
```

- MD_STRUCTURE_NETWORK (3):

  In this query it has to be proofed if there is a member, having more than one parent. For this reason the result is grouped by ?HIERARCHY_UNIQUE_NAME and ?STRUCTURE with the having condition that the count of parents is greater than one.

```
Select distinct
#Structure of concept scheme  = MD_STRUCTURE_NETWORK (3)
?HIERARCHY_UNIQUE_NAME ?STRUCTURE
where{
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
?relationship rdf:type skosclass:ConceptAssociation.
?relationship skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?relationship skosclass:hasSourceConcept ?MEMBER_UNIQUE_NAME.
?relationship rdf:predicate skos:broader.
?relationship skosclass:hasTargetConcept ?PARENT_UNIQUE_NAME.

?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:member ?MEMBER_LEVEL.
?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_LEVEL rdf:type skosclass:ClassificationLevel.

?PARENT_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?PARENT_UNIQUE_NAME skos:member ?PARENT_LEVEL.
?PARENT_UNIQUE_NAME rdf:type skos:Concept.
?PARENT_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?PARENT_LEVEL rdf:type skosclass:ClassificationLevel.

BIND("3" AS ?STRUCTURE)
} GROUP BY
?HIERARCHY_UNIQUE_NAME ?MEMBER_UNIQUE_NAME ?STRUCTURE
HAVING(COUNT(?PARENT_UNIQUE_NAME) > 1)
```

There may be special cases, where a concept scheme is in the result of the query for the structure MD_STRUCTURE_NETWORK (3) and in the result of another query. In this case, the structure of the concept scheme is MD_STRUCTURE_NETWORK (3).

The following SPARQL query can be used for the other important variables for the OLAP4J method getHierarchies. In order to have no members in the result but anyway to count the number of members (?HIERARCHY_CARDINALITY), the result of the query is grouped by the important other variables. Since there may be several labels for a concept scheme (?HIERARCHY_UNIQUE_NAME), it is filtered by the English label. Distinct is needed to guarantee that the same line is not included several times in the result, because the same triple could exist several times. For example if the same English label is crawled several times, because it is included in several information resources, only one of these same triples is needed.

```
Select distinct
#OLAP4J method getHierarchies
?CUBE_NAME
?DIMENSION_UNIQUE_NAME
?HIERARCHY_NAME
?HIERARCHY_UNIQUE_NAME
(count(?MEMBER_UNIQUE_NAME) AS ?HIERARCHY_CARDINALITY)
where{
?CUBE_NAME qb:component ?compSpec.
?CUBE_NAME rdf:type qb:DataStructureDefinition.
?compSpec qb:dimension ?DIMENSION_UNIQUE_NAME.
?DIMENSION_UNIQUE_NAME qb:codeList ?HIERARCHY_UNIQUE_NAME.
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
OPTIONAL{?HIERARCHY_UNIQUE_NAME rdfs:label ?HIERARCHY_NAME.
Filter(lang(?HIERARCHY_NAME)="en")}

?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:member ?LEVEL_UNIQUE_NAME.

?LEVEL_UNIQUE_NAME rdf:type skosclass:ClassificationLevel.
?LEVEL_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
}
GROUP BY
?CUBE_NAME
?DIMENSION_UNIQUE_NAME
?HIERARCHY_NAME
?HIERARCHY_UNIQUE_NAME
```

In the result of this query, the structure is not included since it can be determined with the four queries above. For this reason, the structures of the hierarchies have to be determined with the four queries above and added to the result of this query.

4.3.2 **getLevels**

The following SPARQL query can be used for the OLAP4J method getLevels. In order to have no members in the result but anyway to count the number of members of the level (?LEVEL_CARDINALITY), the result of the query is grouped by the other important variables. Since there may be several labels for a concept scheme (?HIERARCHY_UNIQUE_NAME) and a level (?LEVEL_UNIQUE_NAME), each time it is filtered by the English label. Distinct is needed to guarantee that the same line is not included several times in the result, because the same triple could exist several times. For example if the same English label is crawled several times, because it is included in several information resources, only one of these same triples is needed.

```
Select distinct
#OLAP4J method getLevels
?CUBE_NAME
?DIMENSION_UNIQUE_NAME
?HIERARCHY_NAME
?HIERARCHY_UNIQUE_NAME
?LEVEL_NAME
?LEVEL_UNIQUE_NAME
?LEVEL_NUMBER
(count(?MEMBER_UNIQUE_NAME) AS ?LEVEL_CARDINALITY)
where{
?CUBE_NAME qb:component ?compSpec.
?CUBE_NAME rdf:type qb:DataStructureDefinition.
?compSpec qb:dimension ?DIMENSION_UNIQUE_NAME.
?DIMENSION_UNIQUE_NAME qb:codeList ?HIERARCHY_UNIQUE_NAME.
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
OPTIONAL{?HIERARCHY_UNIQUE_NAME rdfs:label ?HIERARCHY_NAME.
Filter(lang(?HIERARCHY_NAME)="en")}

?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:member ?LEVEL_UNIQUE_NAME.

?LEVEL_UNIQUE_NAME rdf:type skosclass:ClassificationLevel.
?LEVEL_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.

OPTIONAL{?LEVEL_UNIQUE_NAME rdfs:label ?LEVEL_NAME.
Filter(lang(?LEVEL_NAME)="en")}
?LEVEL_UNIQUE_NAME skosclass:depth ?LEVEL_NUMBER.
}
GROUP BY
?CUBE_NAME
?DIMENSION_UNIQUE_NAME
?HIERARCHY_NAME
?HIERARCHY_UNIQUE_NAME
?LEVEL_NAME
?LEVEL_UNIQUE_NAME
?LEVEL_NUMBER
```

4.3.3 **getMembers**

The following SPARQL query can be used for the OLAP4J method getMembers. Hereby, the first part of the where clause which is before the UNION statement selects all members that are part of a concept association and the corresponding relationships. The second part of the where clause which is after the UNION statement selects all members of the uppermost level, since they have no parents. Since a member (?MEMBER_NAME) may have several labels, it is filtered for the English label. Distinct is needed to guarantee that the same line is not included several times in the result, because the same triple could exist several times. For

example if the same English label is crawled several times, because it is included in several information resources, only one of these same triples is needed.

```
Select distinct
#OLAP4J method getMembers
?CUBE_NAME
?DIMENSION_UNIQUE_NAME
?HIERARCHY_UNIQUE_NAME
?LEVEL_UNIQUE_NAME
?LEVEL_NUMBER
?MEMBER_NAME
?MEMBER_UNIQUE_NAME
?PARENT_LEVEL
?PARENT_UNIQUE_NAME
where {
{?CUBE_NAME qb:component ?compSpec.
?CUBE_NAME rdf:type qb:DataStructureDefinition.
?compSpec qb:dimension ?DIMENSION_UNIQUE_NAME.
?DIMENSION_UNIQUE_NAME qb:codeList ?HIERARCHY_UNIQUE_NAME.
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.

?MEMBER_UNIQUE_NAME skos:member ?LEVEL_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.

?LEVEL_UNIQUE_NAME rdf:type skosclass:ClassificationLevel.
OPTIONAl{?MEMBER_UNIQUE_NAME rdfs:label ?MEMBER_NAME.
Filter(lang(?MEMBER_NAME)="en")}
?LEVEL_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?LEVEL_UNIQUE_NAME skosclass:depth ?LEVEL_NUMBER.

?relationship rdf:type skosclass:ConceptAssociation.
?relationship skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?relationship skosclass:hasSourceConcept ?MEMBER_UNIQUE_NAME.
?relationship rdf:predicate skos:broader.
?relationship skosclass:hasTargetConcept ?PARENT_UNIQUE_NAME.

?PARENT_UNIQUE_NAME skos:member ?PARENT_LEVEL.
?PARENT_UNIQUE_NAME rdf:type skos:Concept.
?PARENT_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?PARENT_LEVEL skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?PARENT_LEVEL rdf:type skosclass:ClassificationLevel.
}
UNION
{
?CUBE_NAME qb:component ?compSpec.
?CUBE_NAME rdf:type qb:DataStructureDefinition.
?compSpec qb:dimension ?DIMENSION_UNIQUE_NAME.
?DIMENSION_UNIQUE_NAME qb:codeList ?HIERARCHY_UNIQUE_NAME.
?HIERARCHY_UNIQUE_NAME rdf:type skos:ConceptScheme.
?MEMBER_UNIQUE_NAME skos:member ?LEVEL_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?MEMBER_UNIQUE_NAME rdf:type skos:Concept.
```

```
?LEVEL_UNIQUE_NAME rdf:type skosclass:ClassificationLevel.
OPTIONAl{?MEMBER_UNIQUE_NAME rdfs:label ?MEMBER_NAME.
Filter(lang(?MEMBER_NAME)="en")}
?LEVEL_UNIQUE_NAME skos:inScheme ?HIERARCHY_UNIQUE_NAME.
?LEVEL_UNIQUE_NAME skosclass:depth ?LEVEL_NUMBER.
FILTER(xsd:integer(?LEVEL_NUMBER) =  1)
}}
```

5 Approaches for learning OLAP hierarchies from RDF

In this section the different approaches to generate useful hierarchies from Statistical Linked Data are explained in detail. There have been developed specific approaches for often occurring data (e.g. temporal and geographical) as well as generic approaches that can be applied to all data. Before describing the approaches itself, the steps are explained independently of a concrete approach and ideas for a possible web service are introduced.

An intensive literature study concerning the generation of OLAP hierarchies from Statistical Linked Data has been made, but no algorithms or concepts have been found. Indeed, the developed approaches of this master thesis are not additional methods or extensions of already existing algorithms and can therefore be seen as new ideas to generate OLAP hierarchies. The essential precondition for all approaches is the existence of triples, in which members of a dimension of a data set are linked to other resources which can be used for constructing hierarchies.

To each approach the following information is given, which respectively results in an own subsection. For reasons of clarity and comprehensibility, the naming of URIs is sometimes simplified when describing the approaches:

- An **overview** of the approach
- The possible **parameters**
- The **algorithm** including possible SPARQL construct queries
- An **example** on real data
- The possible **resulting hierarchies**
- Some **criticism**

5.1 Steps of the approaches

The following flow chart shows the steps of generating hierarchies which is independent of a concrete approach. This means that all approaches follow this universally valid schema.

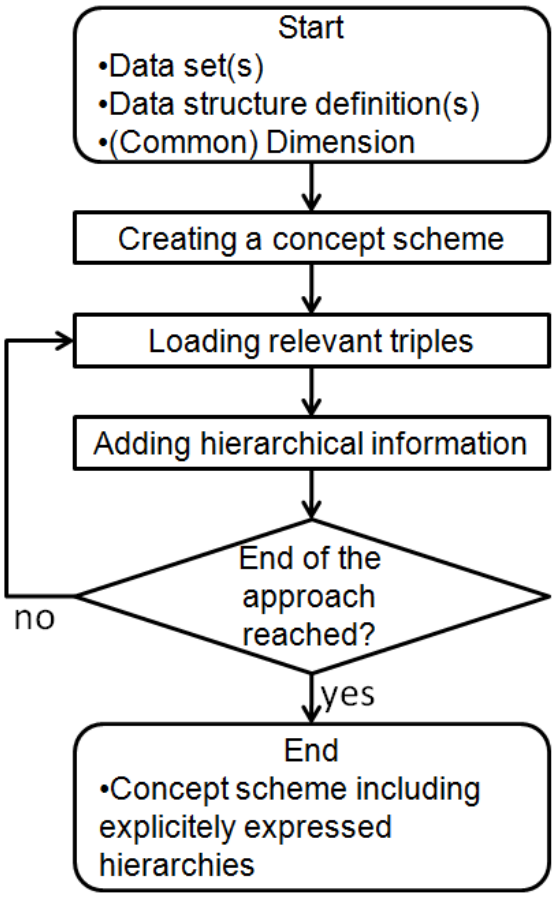

Figure 28: Steps of the approaches

5.1.1 **Start**

For the start of each approach, the following elements have to be given. They all are crawled in a triple store, where the construction of the hierarchies takes place. Also the dictionary information, such as the resources that are owl:sameAs to the members of the given dimension, is loaded in the triple store:

- One or more data sets
- One or more corresponding data structure definitions
- One (common) dimension

5.1.2 **Creating a concept scheme**

The given dimension may be linked via qb:codeList to one or more concept schemes. If there is a concept scheme, which includes concepts for exactly those members that are used in the given dimension of the data set(s), this concept scheme is used.

Otherwise if the concept scheme comprises more or less concepts than members actually are used in the data set(s), a new concept scheme is created, which is linked via the property qb:codeList to the given dimension. For all members of all data sets, including the given dimension, concepts are created, which are linked via rdfs:seeAlso or rdfs:label to the actual member and are assigned via the property skos:inScheme to the created concept scheme. The resulting triples are added to the triple store. The created concept schemes are specific for the given data set(s), but are linked via qb:codeList to the given dimension of the given data structure definition(s).

5.1.3 **Loading relevant triples**

To generate hierarchies out of the created concepts, further metadata of the members is needed. Therefore, relevant triples are loaded and added to the triple store. Loading means both, retrieving triples from a certain information resource and crawling which could for example be done with ldspider[24], a Java implementation for a linked data web crawler. Potentially this could also involve RDF-izing sources, meaning that non-RDF data could also be crawled from relevant data sources which is then transformed into RDF. However, this additional possibility to RDF-izing data isn't used in the approaches of this master thesis.

5.1.4 **Adding hierarchical information**

The crawled triples are used to generate hierarchical information, meaning classification levels, to which concepts are assigned via skos:member and concept associations, to which concepts are assigned via skosclass:hasSourceConcept and skosclass:hasTargetConcept. Both, classification levels and concept associations are assigned to the concept scheme via skos:inScheme. Additional concepts could possibly be created that were also assigned to levels and the concept scheme. Technically speaking, this can be done with SPARQL construct queries in the triple store. The resulting triples are then added to the triple store.

[24] http://code.google.com/p/ldspider/

5.1.5 **End of the approach reached?**

After hierarchical information is added to the concept scheme, it has to be checked, if the end of the approach is reached. If this is the case, no additional triples are crawled and the run of the approach terminates. Otherwise, the run of the approach continues with step 2 ‘Loading relevant triples’.

5.1.6 **End**

If the end of the approach is reached, the created concept scheme represents the resulting hierarchy of the run of the approach. It includes the hierarchical structures, expressed with the introduced SKOS vocabulary and proposed extensions. The resulting concept schemes can then be used in OLAP applications.

Several data sets, including data sets on different aggregation layer, can potential be integrated, using the resulting concept scheme for the common dimension of the data structure definitions. Integration means that observations of different data sets can be combined. The following cases can be distinguished:

- Some dimension members of different data sets have the same parent in a higher level in the created concept scheme. For example for the time dimension the member 2011-12 is used in one data set and the member 2011-08 is used in another data set. Both data sets can be integrated, using the created concept scheme, which includes the parent 2011 of both members.
- Some dimension members are the parents in a higher level of other members of the different data sets in the created concept scheme. For example for the time dimension the member 2011 is used in one data set and the member 2011-08 is used in another data set. Both data sets can be integrated, using the created concept scheme, where 2011 is the parent of 2011-08

5.2 Technical implementation

The technical implementation of the developed approaches to generate hierarchies could either be done as an own system or web based. The idea of this master thesis is to provide a web service which generates hierarchies from Statistical Linked Data.

The different approaches could be represented as request-response port types of a web service. This means that per each approach one method is provided, which can be called by the service consumer. The following URL parameters of a request have to be filled by the service consumer:

- ds: URI(s) of one or more data sets
- dsd: URI(s) of one or more corresponding data structure definitions
- dim: URI of one (common) dimension
- (further approach-specific parameters)

The web service then makes use of the developed approaches depending on the called methods and values of the parameters. Each approach works for the specified dimension, the given data set(s) and the corresponding data structure definition(s).

The response of the service provider is a concept scheme that is linked to the given dimension, including the generated hierarchical structures. It is expressed with the introduced SKOS vocabulary and proposed extensions, serialized in RDF/XML.

5.3 Specific approaches

The specific approaches are developed for the temporal and geographical dimension of a cube. This means that special preconditions have to be fulfilled that these approaches can be used. To each approach, SPARQL construct queries are given, which can be executed to construct the required triples.

5.3.1 **Approach ‘time’**

5.3.1.1 Overview

This approach, called ‘time’, is a special approach for constructing hierarchies within the time dimension in a cube. As postulated by Inmon, data in a data warehouse

should be analyzable within the time dimension, called ‘Time-variant’ [Inmo02, p.31]. The time dimension is contained in many cubes due to the fact that time is always running and measured data is changing over the time. Because of this special character of the time dimension, it is possible to generate hierarchies statically with metadata of a central time service as explained in the following.

Regarding the time dimension there may be specific hierarchies, for example types of days in a certain country (including members: working day, weekend, public holiday) or a university calendar as shown in the following figure:

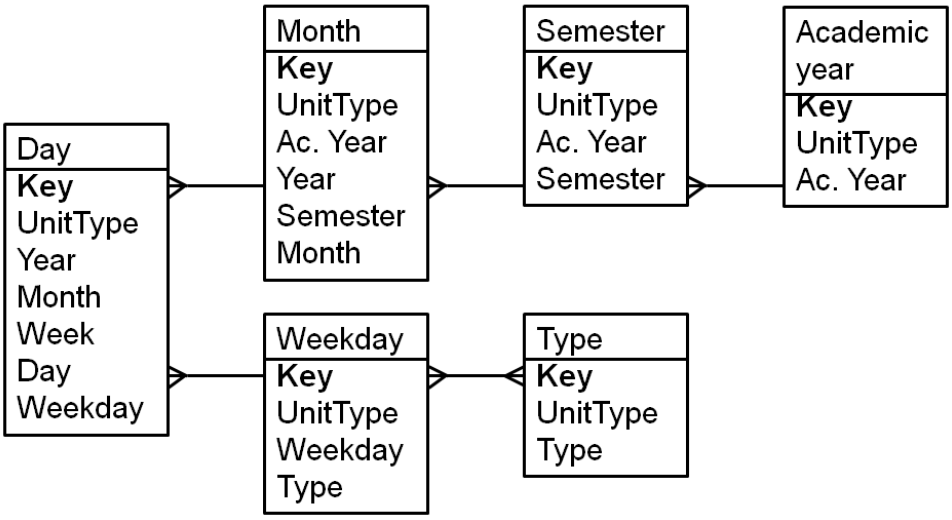

Figure 29: Examples for specific hierarchies of the time dimension: schema

Furthermore, there may also be hierarchies which are universally valid for time points in a specific calendar, e.g. the Gregorian calendar, which is the idea of this approach. The following example of a multiple alternative hierarchy with the common levels ‘Day’ and ‘Year’ serves as running example for this approach.

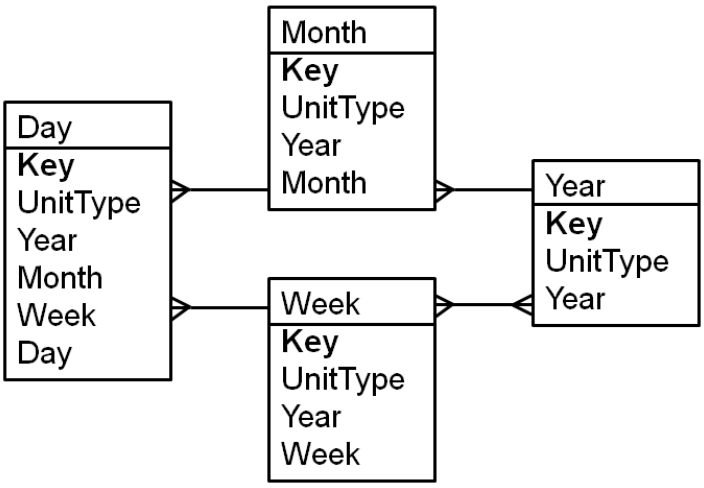

Figure 30: Resulting multiple alternative hierarchy of the approach 'Time': schema

Out of this multiple alternative hierarchy, the following two simple hierarchies that are both symmetric are constructed:

- 'DayToYearViaMonth': This hierarchy is strict and consists of the following three levels:
  Day→Month→Year
- 'DayToYearViaWeek': This hierarchy consists of the three levels
  Day→Week→Year
  and is non-strict since a week can belong to two years. For example the calendar week 01/2012 begins with 12/31/2011 and ends with 06/01/2012, so it belongs to the two years 2011 and 2012, because in a roll up to the level year, the day 12/31/2011 would roll up to the year 2011 and the day 06/01/2012 would roll up to the year 2012 even if they are in the same calendar week.

The assumption made in this example is that a day is the most granular value in statistical data, so the lowest level of the two hierarchies is always a day. For example this can be seen in the time dimension in an EUROSTAT data structure definition[25], which also has as rdfs:range the datatype xsd:date, indicating that a day is the most granular value. However, this approach would also work for more granular data, e.g. on hour, minute and second level. Such a granularity would

[25] http://estatwrap.ontologycentral.com/dsd/teilm020

possibly be required in technical measured data. Therefore the hierarchies could be extended as follows:

Second→Minute→Hour→Day

Although it is assumed that the most general values in statistical data are years, this approach should also work for more general data, e.g. decades, centuries and millenniums. Hence the hierarchies could also be extended on the other side as follows:

Year→Decade→Century→Millennium

Finally, the hierarchies could also be extended by using more levels which are between the month and the year level, e.g. quarters and half-years. So the hierarchies could also be extended in the middle as follows:

Month→Quarter→Half-Year→Year

5.3.1.2 Algorithm

To generate hierarchies out of the time dimension the following steps have to be done.

5.3.1.2.1 Start

The algorithm to generate URIs out of literal values works if the range of a qb:DimensionProperty or the data types of the members within a dimension have one of the following primitive XML Schema data types. If this is not the case, no hierarchies are generated:

- xsd:dateTime
- xsd:date
- xsd:gYearMonth
- xsd:gYear

5.3.1.2.2 Creating a concept scheme

The main idea of this approach is using URIs instead of literals for the time dimension in statistical data sets. Thus, for all literal values that may potentially occur in the time dimension URIs are created. Due to the fact that explicit hierarchies can only be expressed with URIs it is not possible to use literals instead. Consequently, literal values cannot be assigned to concept schemes despite concept schemes for each dimension are needed to express hierarchies. Since only a limited number of different values in the time dimension can occur therefore only a limited number of URIs has to be created.

The lexical representations of the data types are interpreted following the XML Schema specification[26]. For each literal value that occurs as different second, minute, hour, day, month or year an URI is used for representing the corresponding time instants. For reuse reasons the URIs of already existing time services are used. Because of the world-wide use of the Gregorian calendar, this approach also uses the Gregorian calendar, independent of time zones, for what the Gregorian URI set[27] of data.gov.uk[28] is used. This URI set already includes temporal relationships between different time instants which is later used to define hierarchies. In fact, also more concept schemes could be provided using other calendars (e.g. Julian calender) or time-zone specific concept schemes for what the URIs of placetime[29] could be used.

The following table shows the lexical representations for the data types including an example value which results in an example URI. For example the value 2011-10-17 of the xsd:date data type would result in the URI refgovukday:2011-10-17 representing the 17th of October, 2011.

[26] http://www.w3.org/TR/xmlschema-2/
[27] http://www.epimorphics.com/web/wiki/using-interval-set-uris-statistical-data
[28] http://data.gov.uk/
[29] http://www.placetime.com/

Datatype	Lexical representation	Resulting URI
xsd:dateTime	'-'? yyyy '-' mm '-' dd 'T' hh ':' mm ':' ss ('.' s+)? (zzzzzz)? e.g.: 2011-10-17T14:23:17	refgovukminute:2011-10-17T14:23:17
xsd:date	'-'? yyyy '-' mm '-' dd zzzzzz? e.g.: 2011-10-17	refgovukday:2011-10-17
xsd:gYearMonth	'-'? yyyy '-' mm ' e.g.: 2011-10	refgovukmonth:2011-10
xsd:gYear	'-'? yyyy e.g.: 2011	refgovukyear:2011

Table 7: Temporal data types

After the generation of URIs for each literal value these URIs can be linked via the rdfs:label property to the corresponding literal values that are members of the dimension as follows:

refgovukday:2011-10-17	rdfs:label	2011-10-17.

To assign the created URIs to a concept scheme for each of the above described simple hierarchies a concept scheme has to be defined as follows. The label for the concept scheme could be ‘Day to year via month’ respectively ‘Day to year via week’.

hrc:cs/DayToYearViaMonth	rdf:type	skos:ConceptScheme;
	rdfs:label	“Day to year via month”.
hrc:cs/DayToYearViaWeek	rdf:type	skos:ConceptScheme;
	rdfs:label	“Day to year via week”.

The dimension of a data structure definition is then linked via qb:codeList to the created concept schemes if the range of this dimension is one of the above described data types. It has to be noticed that the created concept schemes are specific for a certain data set but are linked via qb:codeList to a dimension of a data structure definition.

The created URIs are then defined as concepts and assigned to the respective schemes as follows. Since the created URIs for days and years are assigned to both concept schemes the created URIs for months are only assigned to the concept schemes ‘DayToYearViaMonth’, since only this concept scheme includes months :

refgovukday:2011-10-17	rdf:type	skos:Concept;
	skos:inScheme	hrc:cs/DayToYearViaMonth;
	skos:inScheme	hrc:cs/DayToYearViaWeek.
refgovukmonth:2011-10	rdf:type	skos:Concept;
	skos:inScheme	hrc:cs/DayToYearViaMonth.
refgovukyear:2011	rdf:type	skos:Concept;
	skos:inScheme	hrc:cs/DayToYearViaMonth;
	skos:inScheme	hrc:cs/DayToYearViaWeek.

For constructing these triples the following SPARQL construct query can be used. Hereby, the given data set(s), data structure definition(s) and dimension have to be filled in the filter statements and an URI for the resulting concept scheme has to be defined and filled in the bind statements (bold marked).

```
Construct {
#Generating concepts for time literals and assigning them to a concept scheme
?dimension qb:codeList ?conceptScheme.
?conceptScheme rdf:type skos:ConceptScheme.
?conceptScheme rdfs:label   “Day to year via month”@en.

?yearConcept rdfs:label ?yearMember.
?yearConcept skos:inScheme ?conceptScheme.
?yearConcept rdf:type skos:Concept.

?monthConcept rdfs:label ?monthMember.
?monthConcept skos:inScheme ?conceptScheme.
?monthConcept rdf:type skos:Concept.

?dayConcept rdfs:label ?dayMember.
?dayConcept skos:inScheme ?conceptScheme.
?dayConcept rdf:type skos:Concept.
}
where{
{
{Select distinct ?conceptScheme ?dimension
```

```
?dayConcept ?dayMember
where {
?dataset rdf:type qb:DataSet.
?dataset qb:structure ?dsd.
?dsd rdf:type qb:DataStructureDefinition.
?dsd qb:component ?component.
?component qb:dimension ?dimension.
{{?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#date>.} UNION {?dimension
rdfs:range <http://www.w3.org/2001/XMLSchema#dateTime>}}
?obs rdf:type qb:Observation.
?obs qb:dataSet ?dataset.
?obs ?dimension ?member.
BIND(URI("conceptSchemeURI") AS ?conceptScheme).
BIND(URI(CONCAT("http://reference.data.gov.uk/id/gregorian-day/", substr(?member,1,10)))
AS ?dayConcept).
BIND(substr(?member,1,10) AS ?dayMember).
FILTER(STRLEN(?member) > 9)
FILTER(
?dataset = <datasetURI1> ||
?dataset = <datasetURI2> ||
…
?dataset = <datasetURIn>
)
FILTER(
?dsd = <dsdURI1> ||
?dsd = <dsdURI2> ||
…
?dsd = <dsdURIn>
)
FILTER(?dimension = <dimensionURI>)
}}
UNION
{Select distinct ?conceptScheme ?dimension
?monthConcept ?monthMember
where {
?dataset rdf:type qb:DataSet.
?dataset qb:structure ?dsd.
?dsd rdf:type qb:DataStructureDefinition.
?dsd qb:component ?component.
?component qb:dimension ?dimension.
{{?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#date>.} UNION {?dimension
rdfs:range <http://www.w3.org/2001/XMLSchema#dateTime>} UNION {?dimension
rdfs:range <http://www.w3.org/2001/XMLSchema#gYearMonth>}}
?obs rdf:type qb:Observation.
?obs qb:dataSet ?dataset.
?obs ?dimension ?member.
BIND(URI("conceptSchemeURI") AS ?conceptScheme).
```

```
BIND(URI(CONCAT("http://reference.data.gov.uk/id/gregorian-month/", ?member)) AS
?monthConcept).
BIND(?member AS ?monthMember).
FILTER(STRLEN(?member) = 7)
FILTER(
?dataset = <datasetURI1> ||
?dataset = <datasetURI2> ||
…
?dataset = <datasetURIn>
)
FILTER(
?dsd = <dsdURI1> ||
?dsd = <dsdURI2> ||
…
?dsd = <dsdURIn>
)
FILTER(?dimension = <dimensionURI>)
}}
UNION
{Select distinct ?conceptScheme ?dimension
?yearConcept ?yearMember
where {
?dataset rdf:type qb:DataSet.
?dataset qb:structure ?dsd.
?dsd rdf:type qb:DataStructureDefinition.
?dsd qb:component ?component.
?component qb:dimension ?dimension.
{{?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#date>.} UNION {?dimension
rdfs:range <http://www.w3.org/2001/XMLSchema#dateTime>} UNION {?dimension
rdfs:range <http://www.w3.org/2001/XMLSchema#gYearMonth>} UNION {?dimension
rdfs:range <http://www.w3.org/2001/XMLSchema#gYear>}}
?obs rdf:type qb:Observation.
?obs qb:dataSet ?dataset.
?obs ?dimension ?member.
BIND(URI("conceptSchemeURI") AS ?conceptScheme).
BIND(URI(CONCAT("http://reference.data.gov.uk/id/gregorian-year/", ?member)) AS
?yearConcept).
BIND(?member AS ?yearMember).
FILTER(STRLEN(?member) = 4)
FILTER(
?dataset = <datasetURI1> ||
?dataset = <datasetURI2> ||
…
?dataset = <datasetURIn>
)
FILTER(
?dsd = <dsdURI1> ||
```

```
?dsd = <dsdURI2> ||
…
?dsd = <dsdURIn>
)
FILTER(?dimension = <dimensionURI>)
}}
}}
```

5.3.1.2.3 Loading relevant triples

This approach uses URIs from data.gov.uk. These triples already include temporal relationships such as interval:intervalContainsMonth. For this reason these triples have to be crawled by the following manner and loaded in the triple store.

Since the needed triples for the temporal relationships are only located in the URIs of the temporal units which include other temporal units these URIs have to be created and they are crawled. If URIs for days are the result of the step before then URIs for months, weeks and years have to be created. If URIs for months are the result of the step before, then URIs for years have to be created. They are crawled and loaded in the triple store. For example if the URI refgovukday:2011-10-17 is given, then the URIs refgovukmonth:2011-10 and refgovukyear:2011 have to be created, crawled and loaded in the triple store. Furthermore, all URIs for weeks in 2011, the URI for the last week in 2010 and the URI for the first week in 2012 have to be created, crawled and loaded in the triple store.

5.3.1.2.4 Adding hierarchical information

After the construction of the concept schemes in step 2, it is required to add the hierarchical information about the included concepts. At first, the hierarchical relationships between the concepts have to be defined and assigned to the respective concept schemes. They are expressed as concept associations which are linked to a source/target concept and to the property skos:broader. For example the concept refgovukday:2011-10-17 indicating the 17th October, 2011 will roll up to the concept refgovukmonth:2011-10 in the simple hierarchy ‘DayToYearViaMonth’ and to the concept refgovukweek:2011-W42 indicating the 42th calendar week in 2011 in the simple hierarchy ‘DayToYearViaWeek’. Both, the concept refgovukmonth:2011-

10 and the concept refgovukweek:2011-W42 itself roll up to the concept refgovukyear:2011 indicating the year 2011.

```
hrc:ca/reference.data.gov.uk/id/gregorian-day:2011-10-17_reference.data.gov.uk/id/
gregorian-month/2011-10   rdf:type                  skosclass:ConceptAssociation;
                          skosclass:hasSourceConcept refgovukday:2011-10-17;
                          rdf:predicate              skos:broader;
                          skosclass:hasTargetConcept refgovukmonth:2011-10;
                          skos:inScheme              hrc:cs/DayToYearViaMonth.
hrc:ca/reference.data.gov.uk/id/gregorian-month/2011-10_reference.data.gov.uk/id/
gregorian-year/2011       rdf:type                  skosclass:ConceptAssociation;
                          skosclass:hasSourceConcept refgovukmonth:2011-10;
                          rdf:predicate              skos:broader;
                          skosclass:hasTargetConcept refgovukyear:2011;
                          skos:inScheme              hrc:cs/DayToYearViaMonth.
hrc:ca/reference.data.gov.uk/id/gregorian-day:2011-10-17_reference.data.gov.uk/id/
gregorian-week/2011-W42   rdf:type                  skosclass:ConceptAssociation;
                          skosclass:hasSourceConcept refgovukday:2011-10-17;
                          rdf:predicate              skos:broader;
                          skosclass:hasTargetConcept refgovukweek:2011-W42;
                          skos:inScheme              hrc:cs/DayToYearViaWeek.
hrc:ca/reference.data.gov.uk/id/gregorian-week/2011-W42_reference.data.gov.uk/id/
gregorian-year/2011       rdf:type                  skosclass:ConceptAssociation;
                          skosclass:hasSourceConcept refgovukweek:2011-W42;
                          rdf:predicate              skos:broader;
                          skosclass:hasTargetConcept refgovukyear:2011;
                          skos:inScheme              hrc:cs/DayToYearViaWeek.
```

Since concepts for those time values that can be found in the data are always part of the concept scheme there may be parent concepts of these (range of skosclass:hasTargetConcept) that are yet not part of the concept scheme. These parents also have to be defined as concepts, linked to the literal value and assigned to the concept scheme. For example the literal value ‘2011-10-17’ is found in the data for what the URI refgovukday:2011-10-17 is created and the literals ‘2011-10’ and ‘2011’ are not found in the data. This means that the URIs refgovukmonth:2011-10

and refgovukyear:2011 have to be created, defined as concepts, linked to its labels and assigned to the concept scheme.

Furthermore, information about levels has to be added to the concept scheme. Therefore, separate levels that are part of the respective concept scheme have to be defined for each temporal unit and assigned to the respective schemes. Also the depth has to be assigned to a certain level as follows:

hrc:cl/day	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	hrc:cs/DayToYearViaMonth;
	skos:inScheme	hrc:cs/DayToYearViaWeek;
	skosclass:depth	“3”.
hrc:cl/month	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	hrc:cs/DayToYearViaMonth;
	skosclass:depth	“2”.
hrc:cl/week	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	hrc:cs/DayToYearViaWeek;
	skosclass:depth	“2”.
hrc:cl/year	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	hrc:cs/DayToYearViaMonth;
	skos:inScheme	hrc:cs/DayToYearViaWeek;
	skosclass:depth	“1”.

The created URIs are then assigned to the respective levels as follows:

refgovukday:2011-10-17	skos:member	hrc:cl/day.
refgovukmonth:2011-10	skos:member	hrc:cl/month.
refgovukweek:2011-W42	skos:member	hrc:cl/week.
refgovukyear:2011	skos:member	hrc:cl/year.

To add this hierarchical information, the following SPARQL construct query can be used. With this query, specific concept schemes corresponding to the simple hierarchy ‘DayToYearViaMonth’ are constructed. For concept schemes corresponding to the simple hierarchy ‘DayToYearViaWeek’ another query is needed. The naming of URIs for levels is depending on the URIs for the concept scheme. The resulting triples are also added to the triple store. If the hierarchical information should only be added to certain concept schemes, it has to be filtered by these concept schemes.

```
Construct {
#Adding hierarchical information to the concept scheme
?yearConcept skos:member ?yearlevel.
?yearConcept skos:inScheme ?conceptScheme.
?yearConcept rdf:type skos:Concept.
?yearConcept rdfs:label ?yearLabel.
?yearlevel skos:inScheme ?conceptScheme.
?yearlevel rdf:type skosclass:ClassificationLevel.
?yearlevel skosclass:depth "1".
?yearlevel rdfs:label "Year"@en.

?monthConcept skos:member ?monthlevel.
?monthConcept skos:inScheme ?conceptScheme.
?monthConcept rdf:type skos:Concept.
?monthConcept rdfs:label ?monthLabel.
?monthlevel skos:inScheme ?conceptScheme.
?monthlevel rdf:type skosclass:ClassificationLevel.
?monthlevel skosclass:depth "2".
?monthlevel rdfs:label "Month"@en.

?dayConcept skos:member ?daylevel.
?dayConcept skos:inScheme ?conceptScheme.
?dayConcept rdf:type skos:Concept.
?dayConcept rdfs:label ?dayLabel.
?daylevel skos:inScheme ?conceptScheme.
?daylevel rdf:type skosclass:ClassificationLevel.
?daylevel skosclass:depth "3".
?daylevel rdfs:label "Day"@en.

?relMonthYear rdf:type skosclass:ConceptAssociation.
?relMonthYear skosclass:hasSourceConcept ?monthConcept.
?relMonthYear rdf:predicate skos:broader.
?relMonthYear skosclass:hasTargetConcept ?yearConcept.
?relMonthYear skos:inScheme ?conceptScheme.

?relDayMonth rdf:type skosclass:ConceptAssociation.
?relDayMonth skosclass:hasSourceConcept ?dayConcept.
?relDayMonth rdf:predicate skos:broader.
?relDayMonth skosclass:hasTargetConcept ?monthConcept.
?relDayMonth skos:inScheme ?conceptScheme.
}
where

{
Select distinct ?conceptScheme
?yearConcept ?yearlevel ?yearLabel
?monthConcept ?monthlevel ?monthLabel
```

```
?dayConcept ?daylevel ?dayLabel
?relMonthYear ?relDayMonth
where {
?dsd rdf:type qb:DataStructureDefinition.
?dsd qb:component ?component.
?component qb:dimension ?dimension.

{{?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#date>.} UNION {?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#dateTime>} UNION {?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#gYearMonth>} UNION {?dimension rdfs:range <http://www.w3.org/2001/XMLSchema#gYear>}}
?dimension qb:codeList ?conceptScheme.

{{?yearConcept skos:inScheme ?conceptScheme.
?yearConcept rdf:type skos:Concept.
?yearConcept rdf:type interval:Year.
}
UNION
{?monthConcept skos:inScheme ?conceptScheme.
?monthConcept rdf:type skos:Concept.
?monthConcept rdf:type interval:Month.
?yearConcept interval:intervalContainsMonth ?monthConcept.
?yearConcept rdf:type interval:Year.
}
UNION
{?dayConcept skos:inScheme ?conceptScheme.
?dayConcept rdf:type skos:Concept.
?dayConcept rdf:type interval:Day.
?monthConcept interval:intervalContainsDay ?dayConcept.
?monthConcept rdf:type interval:Month.
?yearConcept interval:intervalContainsMonth ?monthConcept.
?yearConcept rdf:type interval:Year.
}}

BIND(substr(str(?yearConcept),48) AS ?yearLabel).
BIND(substr(str(?monthConcept),49) AS ?monthLabel).
BIND(substr(str(?dayConcept),47) AS ?dayLabel).

BIND(URI(CONCAT("http://hierarchie.org/cl/time/year/",
substr(str(REPLACE(str(?conceptScheme),"\\.","/")),26))) AS ?yearlevel).
BIND(URI(CONCAT("http://hierarchie.org/cl/time/month/",
substr(str(REPLACE(str(?conceptScheme),"\\.","/")),26))) AS ?monthlevel).
BIND(URI(CONCAT("http://hierarchie.org/cl/time/day/",
substr(str(REPLACE(str(?conceptScheme),"\\.","/")),26))) AS ?daylevel).
```

```
BIND(URI(CONCAT("http://hierarchie.org/ca/",
substr(str(REPLACE(str(?monthConcept),"\\.","/")),8),"_",substr(str(REPLACE(str(?yearConc
ept),"\\.","/")),8) )) AS ?relMonthYear ).
BIND(URI(CONCAT("http://hierarchie.org/ca/",
substr(str(REPLACE(str(?dayConcept),"\\.","/")),8),"_",substr(str(REPLACE(str(?monthConce
pt),"\\.","/")),8) )) AS ?relDayMonth ).
}}
```

5.3.1.2.5 End of the approach reached?

This approach is straight forward which means that there are no iterations, because the needed URIs have already been created and crawled, since only the URIs of the higher temporal unit include the temporal relationships. For example the URIs for months include the triples for the relationships of months to the included days.

5.3.1.2.6 End

In the end of the approach 'time' the constructed concept scheme includes the hierarchical relationships which can then be used.

5.3.1.3 Example

There are two data sets[30] [31] which have the common dimension dcterms:date. The range of this dimension is defined as xsd:date in the corresponding data structure definitions[32] [33] and the following values occur in this dimension:

Data set	Values
http://estatwrap.ontologycentral.com/id/teilm020#ds	2011-04 till 2011-11
http://estatwrap.ontologycentral.com/id/tsieb020#ds	2006 till 2013

Before the above queries, which generate concept schemes for the time dimension with the levels day, month and year can be executed, the filters for the these specific data sets (SPARQL-variable ?dataset), the corresponding data structure definitions

[30] http://estatwrap.ontologycentral.com/id/teilm020#ds
[31] http://estatwrap.ontologycentral.com/id/tsieb020#ds
[32] http://estatwrap.ontologycentral.com/dsd/teilm020#dsd
[33] http://estatwrap.ontologycentral.com/dsd/tsieb020#dsd

(SPARQL-variable ?dsd) and the dimension (SPARQL-variable ?dimension) have to be set in the first query. After executing both queries, there are generated concepts for the months 2011-04 till 2011-11 and years 2006 till 2013, since only these values occur in the dimension.

5.3.1.4 Resulting hierarchies

All resulting hierarchies are simple hierarchies. They can be symmetric or asymmetric which depends on the occurring values of the time dimension. All concept schemes that include levels for months are strict. The resulting concept schemes which include levels for weeks could possibly be non-strict, if they include the last week of a calendar year, which rolls up to two years.

5.3.1.5 Criticism

Although this approach can probably be applied to many data sets, because many data sets include a temporal dimension this approach can only be used for these dimensions. So this approach has to be seen as a specific and not generic approach.

If a time dimension is used in several data structure definitions, there are created several data set specific concept schemes, one for each data set. This means that only concepts for the occurring values in data sets and their parents till to the root level are part of a created concept scheme and not all theoretical possible values. If a data set is analyzed, the corresponding concept scheme has to be chosen. If various data sets are analyzed together, the different concept schemes can be combined by setting the particular levels and the concept schemes owl:sameAs.

Because the time dimension is contained in many cubes, it would also make sense to provide concept schemes including all days, months, weeks and years between two particular time instants, for example for all days, months, weeks and years beginning with the 1st January, 1900 and for now ending with the 31st December, 2099. The advantage of such universal valid concept schemes is that several data sets could be integrated without setting the created data set specific concept schemes and the included levels owl:sameAs. However, the disadvantage of such universal valid

concept schemes is that many days, months, weeks and years would be needless and would possibly impact negatively the performance.

Since the OWL time ontology[34] is an official W3C working draft the Gregorian URI set[35] of data.gov.uk[36] could also be extended with classes and properties of the OWL time ontology to gain a broad support of this approach. The resources for each time point of the temporal units day, week, month and year have also be defined as DateTimeDescription of the OWL time ontology. Depending on the temporal unit the meaningful properties are generated and filled with values. For example the resource refgovukday:2011-10-17 could be enriched as follows:

refgovukday:2011-10-17	rdf:type	time:DateTimeDescription;
	time:unitType	time:unitMonth;
	time:year	2011;
	time:month	10;
	time:week	42;
	time:day	17.

Since there is no temporal unit in the OWL time ontology designated for quarters or half years the simple hierarchy ‘DayToYearViaMonth’ cannot be extended by these two more levels. For this purpose the central concept scheme has to extended or another concept scheme has to be defined where the concepts and levels for quarters or half years are not in union instances of the OWL time ontology.

Furthermore, temporal relationships of triples in refgovuk are only located in the URIs of the temporal units, which include other temporal units, but not reverse. For example the following triple can be found in the information resource of the URI refgovukyear:2011 but not in the information resource of the URI refgovukmonth:2011-10.

refgovukyear:2011	interval:intervalContainsMonth	refgovukmonth:2011-10

[34] http://www.w3.org/TR/owl-time/
[35] http://www.epimorphics.com/web/wiki/using-interval-set-uris-statistical-data
[36] http://data.gov.uk/

To generate parents of temporal units such triples should also be included in the URIs of the temporal units which are included in other temporal units. For example the triple above should also be found in the information resource of the URI refgovukmonth:2011-10.

5.3.2 **Approach 'geo'**

5.3.2.1 Overview

Besides the temporal relatedness of observations also the geographic relatedness is very often expressed in Statistical Linked Data. For this reason this approach, called 'geo', shows how geographical hierarchies can be constructed.

There are a few vocabularies in the Linked Data Cloud providing geographical data (e.g. geonames[37], freebase[38]). This approach uses the Nomenclature of Territorial Units for Statistics (NUTS)[39] which is a classification service for the territory of the European Union defined by the Eurostat office of the European Union.

NUTS is based on the existing national administrative structures. It categorizes the regions in four levels, although the national administrative structures of the particular countries vary in fact. The aim is to group similar regions together in the same level in order to make comparison and analysis [CGSH10]. Hereby each region in the same level is either the expression of a political will or meant to provide comparable features for statistics. The following table with the not rigidly applied thresholds for inhabitants shows the levels for the established regions of the NUTS classification.

Level	Description	Minimum	Maximum
NUTS 0	Countries	-	-
NUTS 1	States	3 million	7 million
NUTS 2	Administrative regions	800.000	3 million
NUTS 3	Counties/districts and greater metropolitan areas	150.000	800.000

Table 8: Levels of the NUTS classification system

[37] http://www.geonames.org
[38] http://www.freebase.org
[39] http://nuts.geovocab.org

For those countries where the administrative is not structured that way or the size regarding inhabitants of existing regions is too small, additional members are created. They may be assigned to different levels:

- 1st level (eg. no states like ‘Centro’ in Spain)
- 2nd level (eg. no administrative regions like ‘Freiburg’ in Germany)
- 3rd level (eg. three members in Level NUTS 3 for the arrondissement ‘Verviers’ in Belgium)

5.3.2.2 Algorithm

To generate hierarchies out of the geo dimension with help of the hierarchical classification system NUTS the following steps have to be done.

5.3.2.2.1 Start

The algorithm to generate hierarchies with this approach works in two cases. Either the range of the given dimension or the data types of the members within the given dimension of the given data sets have the data type nutsdef:NUTSRegion. If this is not the case, no hierarchies are generated:

5.3.2.2.2 Creating a concept scheme

To assign the created URIs to a concept scheme at first for each of the above described simple hierarchies, a concept scheme has to be defined as follows. The name of the concept scheme could be ‘NUTS.’

hrccs:NUTS	rdf:type	skos:ConceptScheme;
	rdfs:label	“NUTS”@en.

The dimension of a data structure definition is then linked via qb:codeList to the created concept scheme. It has to be noticed that the created concept schemes are specific for a certain data set, but are linked via qb:codeList to a dimension of a data structure definition.

As described above, the members of a dimension in the RDF Data Cube Vocabulary should be separated from the concepts in SKOS. Therefore for each member of the given dimension in the given data sets, a new resource is created. The URIs for the created concepts is a concatenation of ‘http://hierarchie.org/co/NUTS/’ and the NUTS shorthand symbol. For example for the NUTS resource <http://nuts.psi.enakting.org/id/DE>, which is owl:sameAs to the member <http://estatwrap.ontologycentral.com/dic/geo#DE> of a data set the new resource hrcco:NUTS/DE is created, which is then defined as SKOS concept, linked to the original member via rdfs:seeAlso and assigned to the created concept scheme.

hrcco:NUTS/DE	rdf:type	skos:Concept;
	skos:inScheme	hrccs:NUTS;
	rdfs:seeAlso	<http://nuts.psi.enakting.org/id/DE>.

For constructing these triples, the following SPARQL construct query can be used. Hereby, the given data set(s) and dimension have to be filled in the filter statements and an URI for the resulting concept scheme has to be defined and filled in the bind statements (bold marked).

```
Construct{
#Creating a concept scheme for the approach ‘geo’
?concept rdf:type skos:Concept.
?concept rdfs:seeAlso ?nutsRegion.
?concept skos:inScheme ?conceptScheme.
?conceptScheme rdf:type skos:ConceptScheme.
?conceptScheme rdfs:label “NUTS”@en.
?dimension qb:codeList ?conceptScheme.
}
where {
?obs rdf:type qb:Observation.
?obs qb:dataSet ?dataset.
?dataset rdf:type qb:DataSet.
?dataset qb:structure ?dsd.
?obs ?dimension ?member.
BIND(URI(CONCAT("http://hierarchie.org/co/NUTS/", SUBSTR(STR(?member),46))) AS
?concept).
?member owl:sameAs ?nutsRegion.
Filter(STRSTARTS(str(?nutsRegion),"http://nuts.psi.enakting.org"))
FILTER(
?dataset = <datasetURI1> ||
```

```
?dataset = <datasetURI2> ||
…
?dataset = <datasetURIn>
)
FILTER(
?dsd = <dsdURI1> ||
?dsd = <dsdURI2> ||
…
?dsd = <dsdURIn>
)
FILTER(?dimension = <dimensionURI>)
BIND(URI("conceptSchemeURI") AS ?conceptScheme).
}
```

5.3.2.2.3 Loading relevant triples

In this step, triples have to be crawled to retrieve the relationships between the NUTS regions. Before this step is executed, new concepts have been assigned in the step before to the concept scheme. All URIs of NUTS resources that were linked via rdfs:seeAlso to these concepts are crawled and the triples are added to the triple store. This is done to determine the resulting parents of the crawled resources. For example if a concept for the NUTS resource nuts:DE12 has been added to the concept scheme in the step before, then the triples of this URI are crawled. This is for example the following triple, where the subject nuts:DE1 indicates the parent of the concept for the NUTS resource nuts:DE12.

nuts:DE1	spatial:contains	nuts:DE12

5.3.2.2.4 Adding hierarchical information

After the concept schemes have constructed in step 2, hierarchical information about the included concepts has to be added. At first the hierarchical relationships between the concepts have to be defined and assigned to the respective concept schemes. They are expressed as concept associations, linked to a source/target concept and to the property skos:broader. The determination of the relationships between child and parent is done with the help of the property spatial:contains, which is used in NUTS.

hrcca:NUTS/DE1_/DE	rdf:type	skosclass:ConceptAssociation;

	skosclass:hasSourceConcept	hrcco:NUTS/DE1;
	rdf:predicate	skos:broader;
	skosclass:hasTargetConcept	hrcco:NUTS/DE;
	skos:inScheme	hrccs:NUTS.

Since concepts for those members that can be found in the data are always part of the concept scheme, there may be parent concepts of these (range of skosclass:hasTargetConcept) that are yet not part of the concept scheme. These parents also have to be defined as concepts and assigned to the concept scheme.

Furthermore information about levels has to be added to the concept scheme. Therefore separate levels that are part of the respective concept scheme have to be defined for NUTS Level and assigned to the respective schemes. Also the depth has to be assigned to a certain level as follows. For example the level NUTS0 results in the uppermost level which has depth 1. The names for the levels could be a concatenation of 'NUTS' and the NUTS level.

hrccl:NUTS/NUTS0	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	hrccs:NUTS;
	skosclass:depth	"1";
	rdfs:label	„NUTS0"@en.
hrccl: NUTS/NUTS1	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	hrccs:NUTS;
	skosclass:depth	"2";
	rdfs:label	„NUTS1"@en.

The created URIs are then assigned to the respective levels as follows:

hrcco:NUTS/DE	skos:member	hrccl:NUTS/NUTS0
hrcco:NUTS/DE1	skos:member	hrccl:NUTS/NUTS1.

These steps can be performed with the following SPARQL construct query: The naming of URIs for levels is dependent of the URIs for the concept scheme. Hereby,

the URI of the concept scheme has to be filled in (bold marked). The resulting triples are also added to the triple store:

```
Construct {
#Adding hierarchical information to the concept scheme
?relationship skosclass:hasSourceConcept ?childConcept.
?relationship rdf:predicate skos:broader.
?relationship skosclass:hasTargetConcept ?parentConcept.
?relationship rdf:type skosclass:ConceptAssociation.
?relationship skos:inScheme ?conceptScheme.

?childConcept  skos:member ?childLevel.
?parentConcept skos:member ?parentLevel.
?childLevel    rdf:type skosclass:ClassificationLevel.
?parentLevel   rdf:type skosclass:ClassificationLevel.
?childLevel    skos:inScheme ?conceptScheme.
?parentLevel   skos:inScheme ?conceptScheme.
?childLevel    skosclass:depth ?childDepth.
?parentLevel   skosclass:depth ?parentDepth.
?childLevel    rdfs:label ?childLevelLabel.
?parentLevel   rdfs:label ?parentLevelLabel.

?parentConcept rdf:type skos:Concept.
?parentConcept skos:inScheme ?conceptScheme.
?parentConcept rdfs:seeAlso ?parent.
?parentConcept rdfs:label ?parentLabel.

}
where {
OPTIONAL{ ?parent spatial:contains ?child.}
?child  rdf:type nutsdef:NUTSRegion.
?parent rdf:type nutsdef:NUTSRegion.
?child  nutsdef:code ?childCode.
?parent nutsdef:code ?parentCode.
BIND(URI(CONCAT("http://hierarchie.org/ca/NUTS/",      SUBSTR(str(?child),33)      ,"_"      ,
SUBSTR(str(?parent),33))) AS ?relationship).

?childConcept skos:inScheme ?conceptScheme.
Filter(?conceptScheme = <conceptSchemeURI>)
?childConcept rdf:type skos:Concept.
?childConcept rdfs:seeAlso ?child.

BIND(URI(CONCAT("http://hierarchie.org/co/NUTS/",      SUBSTR(STR(?parent),33)))      AS
?parentConcept).

BIND(URI(CONCAT("http://hierarchie.org/cl/NUTS/",substr(REPLACE(str(?conceptScheme),
"\\.","/"),26),SUBSTR(str(datatype(?childCode)),34,5))) AS ?childLevel).
```

```
BIND(URI(CONCAT("http://hierarchie.org/cl/NUTS/",substr(REPLACE(str(?conceptScheme),
"\\.","/"),26),SUBSTR(str(datatype(?parentCode)),34,5))) AS ?parentLevel).
BIND(xsd:integer(SUBSTR(str(datatype(?childCode)),38,1))+1 AS ?childDepth)
BIND(xsd:integer(SUBSTR(str(datatype(?parentCode)),38,1))+1 AS ?parentDepth)
BIND(CONCAT("NUTS",str(xsd:integer(SUBSTR(str(datatype(?childCode)),38,1)))) AS
?childLevelLabel)
BIND(CONCAT("NUTS",str(xsd:integer(SUBSTR(str(datatype(?parentCode)),38,1)))) AS
?parentLevelLabel)
}
```

5.3.2.2.5 End of the approach reached?

The end of the approach is reached, if no new triples have been constructed with the last step. Otherwise the algorithm continues with step 2.

5.3.2.2.6 End

In the end of the approach ‘geo’, the labels of the members have also to be assigned to the concepts, which can be done with the following SPARQL construct query. The URI for the concept scheme has to be inserted (bold marked).

```
Construct {
?concept rdfs:label ?label.
}
where {
?concept rdfs:seeAlso ?member.
?concept rdf:type skos:Concept.
?member rdfs:label ?label.
?concept skos:inScheme ?conceptScheme.
Filter(?conceptScheme = <conceptSchemeURI>)
}
```

The constructed concept scheme includes then the hierarchical relationships which can be used.

5.3.2.3 Example

There are two data sets[40] [41], which have the common dimension eus:geo. The range of this dimension is defined as <http://rdfdata.eionet.europa.eu/ramon/ontology/NUTSRegion> in the corresponding data structure definitions[42] [43] and NUTS regions of the following NUTS levels occur in this dimension as members:

Data set	NUTS levels
http://estatwrap.ontologycentral.com/id/agr_r_landuse#ds	NUTS0, NUTS1, NUTS2
http://estatwrap.ontologycentral.com/id/tsieb020#ds	NUTS0

Table 9: Example for the approach 'geo'

Before the above queries can be executed, the filters for the these specific data sets (SPARQL-variable ?dataset), the corresponding data structure definitions (SPARQL-variable ?dsd) and the dimension (SPARQL-variable ?dimension) have to be set in the first query. After executing both queries, there are generated concepts for the NUTS regions in level NUTS0, NUTS1 and NUTS2 since only these values occur in the dimension.

5.3.2.4 Resulting hierarchies

The resulting hierarchies are all simple hierarchies that are symmetric or asymmetric, depending on the occurring values of the given dimension. Furthermore they are strict, since one NUTS region is included in at most one another NUTS region.

5.3.2.5 Criticism

Besides the time approach, this approach can also be seen as a specific approach, since it doesn't work for generic vocabularies but only for NUTS. So this approach can be applied to all data sets, including geographical dimensions, whose members are linked to NUTS.

[40] http://estatwrap.ontologycentral.com/id/agr_r_landuse#ds
[41] http://estatwrap.ontologycentral.com/id/tsieb020#ds
[42] http://estatwrap.ontologycentral.com/dsd/agr_r_landuse#dsd
[43] http://estatwrap.ontologycentral.com/dsd/tsieb020#dsd

Since NUTS itself is only a classification scheme for Europe, other vocabularies or extensions to NUTS are required for worldwide statistical data, including other parts of the world with additional countries. This means that this approach works only for members of the given dimension that are all defined as NUTS regions. If the given data sets also include members that are not defined as NUTS region, these members have to be assigned to a created level, in case of doubt to the lowest level.

Furthermore NUTS does not consider the administrative structure of the regions at all, because it tries to squeeze all regions in one classification system with four common levels. On the one hand this has positive effects on the comparability between regions. On the other hand the real world is represented incorrectly, since additional members are created. The developed extension of SKOS provides the possibility to consider the structural specialties by using a generalized hierarchy. Therefore different levels for each type of region such as state, administrative district, arrondissement, province, etc. would be required.

NUTS has one specialty, because the arrondissement Verviers (nuts:BE333) in the province Liège (nuts:BE33) in Belgium is split in a German (nuts:BE336) and French (nuts:BE335) speaking part. This is represented in NUTS as 3 members in level NUTS 3 and the following relationships:

nuts:BE33	spatial:contains	nuts:BE333
nuts:BE33	spatial:contains	nuts:BE335
nuts:BE33	spatial:contains	nuts:BE336
nuts:BE333	nutsdef:splitted	nuts:BE335
nuts:BE333	nutsdef:splitted	nuts:BE336

Although the NUTS classification itself and the resulting concept scheme are symmetric, the resulting concept scheme could also be asymmetric, considering this specialty. This means that for example the concept for the French speaking part (hrc:co/nuts.psi.enakting.org/id/BE335) would be assigned to the arrondissement Verviers (hrc:co/nuts.psi.enakting.org/id/BE333) and not directly to the province Liège (hrc:co/nuts.psi.enakting.org/id/BE33), which is the broader concept of Verviers

(hrc:co/nuts.psi.enakting.org/id/BE333). So the members hrc:co/nuts.psi.enakting.org/id/BE335 and hrc:co/nuts.psi.enakting.org/id/BE336 could be assigned to an additional level that is not mandatory for all members, which would result in an asymmetric hierarchy. However, this approach does not consider this specialty.

5.4 Generic approaches

The generic approaches are developed for general use. This means that they are not limited to special dimensions like the approach ‘time’ or ‘geo’. Therefore the possible result set of generated hierarchies can be influenced with some parameters. For these generic approaches it is assumed that the members of a dimension in a given data set are of the same granularity. As a consequence, the members of a dimension are assigned to the same hierarchy level after execution the algorithms of the generic approaches.

5.4.1 **Approach ‘rdfs:subClassOf’**

5.4.1.1 Overview

This approach, called ‚rdfs:subClassOf‘, uses the types (rdf:type) of the dimension members and the relationships of the typed RDF classes (rdfs:subClassOf). Since the property rdf:type is used to state that a resource is an instance of a class, the property rdfs:subClassOf is used to state that all the instances of one class are instances of another [HiKR09, pp. 46-67]. This approach considers only dimension members, which are URIs and not literals, since literals cannot be typed to a class. Essential condition for this approach is the existence of further metadata to each member in a way that hierarchies can be derived.

5.4.1.2 Parameters

This approach is developed for general use. Theoretically, many possible hierarchies can result with this approach. The result set of possibly hierarchies of this approach

can be influenced with the following parameters, which are in union termination criteria for the algorithm:

- Minimal number of stages:
  This parameter defines how many stages a resulting hierarchy has at least.
- Maximal number of stages:
  This parameter defines how many stages a resulting hierarchy has at most.
- Flag for schema:
  If this flag is set, the schema of the ontology is considered. This means that superclasses, subclasses and instances result each time in another element of the hierarchy. If this flag is not set, superclasses, subclasses and instances result each time in a member. The following table gives on overview of this mapping.

Ontology	**Hierarchy**	
	Flag for schema set	**Flag for schema not set**
superclass	level	member
subclass	member of the parent level & level	member
instance	member	member

Table 10: Mapping between ontology and hierarchy of the approach 'rdfs:subClassOf'

- Flag for the strictness:
  This parameter serves as decision criterion for the strictness of the resulting hierarchy. If this flag is set, the resulting hierarchy is strict. This may semantically be not correct, since there may be cases where a resource is an instance of several classes or a class is sub class of several other classes. This would result in several parent members. If this flag is set, one class is chosen to have only one parent member. If non-strictness is required, this flag has not to be set.
- Flag for more members than parents:
  If there would exist more parent members within one level than members, this would rather correspond to a drill down and not to a roll up. Because higher hierarchy levels should rather be reached by a roll up operation, more parents than members would not be reasonable in many cases. So if this flag is set,

there have to exist more members than parent members within one level. If this is not the case, the actual classes are forming no higher level.

5.4.1.3 Algorithm

5.4.1.3.1 Start

It is assumed that the members of a given dimension result in the lowest level of the hierarchy. This means they are the subjects from which this approach starts from. The following steps have then to be done.

5.4.1.3.2 Creating a concept scheme

To assign the created URIs to a concept scheme, for each of the above described simple hierarchies, a concept scheme has to be defined as follows. The given dimension of the given data structure definition(s) is then linked via qb:codeList to it.

hrccs:properties	rdf:type	skos:ConceptScheme.

As described above, the members of a dimension in the RDF Data Cube Vocabulary should be separated from the concepts of the classification hierarchy. Therefore for each member of the given dimension in the given data sets a new URI is created. The new URI can be orientated towards the URI of the member. For example for the URI <http://estatwrap.ontologycentral.com/dic/geo#DE> the new URI hrcco:estatwrap/ontologycentral/com/dic/geo/DE is created, which is then used as SKOS concept and assigned to the created concept scheme. The concept is then linked via rdfs:seeAlso to the original member:

hrcco:estatwrap/ontologycentral/com/dic/geo/DE	rdf:type	skos:Concept;
skos:inScheme	hrccs:subClassOf;	
rdfs:seeAlso	<http://estatwrap.ontologycentral.com/dic/geo#DE>.	

For constructing these triples the following SPARQL construct query can be used. Hereby the given data set(s) and dimension have to be filled in the filter statements

and an URI for the resulting concept scheme has to be defined and filled in the bind statements (bold marked).

```
Construct{
#Creating a concept scheme
?concept rdf:type skos:Concept.
?concept rdfs:seeAlso ?member.
?concept skos:inScheme ?conceptScheme.
?conceptScheme rdf:type skos:ConceptScheme.
?dimension qb:codeList ?conceptScheme.
}

where{
?obs rdf:type qb:Observation.
?obs qb:dataSet ?dataset.
?dataset rdf:type qb:DataSet.
?obs ?dimension ?member.
?dataset qb:structure ?dsd.
BIND(URI(CONCAT("http://hierarchie.org/co/",
REPLACE(REPLACE(SUBSTR(STR(?member),8),"\\.","/"),"#","/"))) AS ?concept).

FILTER(
?dataset = <datasetURI1> ||
?dataset = <datasetURI2> ||
…
?dataset = <datasetURIn>
)
FILTER(
?dsd = <dsdURI1> ||
?dsd = <dsdURI2> ||
…
?dsd = <dsdURIn>
)
FILTER(?dimension = <dimensionURI>)
BIND(URI("conceptSchemeURI") AS ?conceptScheme).
}
```

5.4.1.3.3 Loading relevant triples

For this approach only additional triples are required, which have the following properties. All other triples are not needed, since they are not used:

- rdf:type
- rdfs:subClassOf

- rdfs:label
- owl:equivalentClass
- owl:sameAs

In the first run all classes of the member and the resources that are linked to it with the owl:sameAs property are determined using the rdf:type property. This means triples are crawled from the URI of the member of the given dimension and to owl:sameAs resources of the member.

With each further iteration each further super classes of the present found classes are determined using the rdfs:subClassOf property. Hereby all owl:eqivalentClass relationships are considered. This is done till no more super classes are needed because of the parameter ‘Maximal number of stages’ or no more further super classes are found.

All these triples are added to the triple store.

5.4.1.3.4 Adding hierarchical information

Since this approach can have recursions, there has to be distinguished between the first and the other passes of this step.

At the first pass of this step, a leaf level is created, to which the concepts of the created concept scheme are assigned to via skos:member. This level is assigned to the concept scheme via skos:inScheme.

hrccl:properties/leaf	rdf:type	skosclass:ClassificationLevel;
	skos:inScheme	Properties.
hrcco:estatwrap/ontologycentral/com/dic/geo/DE		
	skos:member	hrccl:properties/leaf.

When generating hierarchies with this approach each redundant instantiation of a super class is ignored so it does not result in a member of a level. This means that each time only the type of the lowest class is relevant, since this additional instantiation of the super class in the ontology could be derived by reasoning.

As a consequence, the hierarchical information can only be added to the concept scheme when all relationships between classes are known. This means that this additional hierarchical information is not added to the concept scheme before not all relevant triples are crawled. Therefore, the recursions serve only for crawling relevant triples and add them to the triple store. Only at the last recursion, triples are generated, which express the hierarchical information.

In the last recursion the generation of hierarchies is as follows. Basically, it has to be distinguished if the parameter ‘Flag for schema’ is set or not, because the process of generating hierarchies is different:

- If the ‘Flag for schema’ is set:
  The basic idea of generating levels is that each class of the ontology results in an own level of the hierarchy. The label of the class in the ontology results in the name of the level in the hierarchy. All subclasses of a class result in the members of the level whereby the labels of the subclasses result in the labels of the members. Also each instance of a class results in a member of the hierarchy, whereby its label results in the label of the member. If this flag is set, the resulting hierarchy is a generalized hierarchy, since levels are split. Because a member can only roll up to one level in a generalized hierarchy, one parent has to be chosen, if levels are split.
- If the ‘Flag for schema’ is not set:
  The basic idea of generating levels is that a class of the ontology results in a member of the hierarchy. Levels are formed by the sequence of rdfs:subClassOf relationships, which means that all classes that are on the same stage within one simple hierarchy are forming one level. Each instance of a class results in a member of the level below the class, whereby its label results in the label of the member.

In each case each uppermost class that is crawled results in an uppermost level, which means that the hierarchy is parallel.

5.4.1.3.5 End of the approach reached?

With each further iteration each further super classes of the present found classes are determined, using the rdfs:subClassOf property. This is done till no more super classes are needed because of the parameter ‘Maximal number of stages’ or no more further super classes are found. Only in the last iteration the hierarchical information is added to the concept scheme, since all classes have to be known. This means that the end of the approach is reached, if hierarchical information has been added to the concept scheme in the step before. Otherwise the algorithm continues with the second step, where additional super classes are determined.

For example, if the value for the parameter ‘Maximal number of stages’ is set to three, there are the following iterations:

1. Determination of all classes of which the members are an instance of. Creating the leaf level an assigning the given members to the leaf level.
2. Determination of further super classes of the already found classes.
   No hierarchical information is added to the concept scheme.
3. Determination of further super classes of the already found classes.
   The hierarchical information is added to the concept scheme, because all needed classes are known.

If the value for the parameter ‘Maximal number of stages’ is increased, the second part would be repeated as often the value for the parameter ‘Maximal number of stages’ increases.

5.4.1.3.6 End

In the end of the approach ‘rdfs:subClassOf’ the constructed concept scheme includes the hierarchical relationships which can then be used.

5.4.1.4 Example

There is a data set[44], which has the dimension eus:geo. The members of this dimension include owl:sameAs links to resources from dbpedia[45], an structured

[44] http://estatwrap.ontologycentral.com/id/tsieb020#ds

extract of wikipedia[46]. The following figure shows the simplified example of some resources and their relationships. The ellipses are denoting the classes, the arrows are indicating a rdfs:subClassOf relationship and the rectangles are denoting the dimension members that are instances of one or more classes. For illustration reasons the following simplifications are made:

- Only six members/instances are used
- Only eight classes are used
- For illustration reasons the class ‘Alliance of states’ is fictional and the subClassOf relationship between the classes ‘Federal Countries’ and ‘Country’ is removed
- The label is used instead of the URI

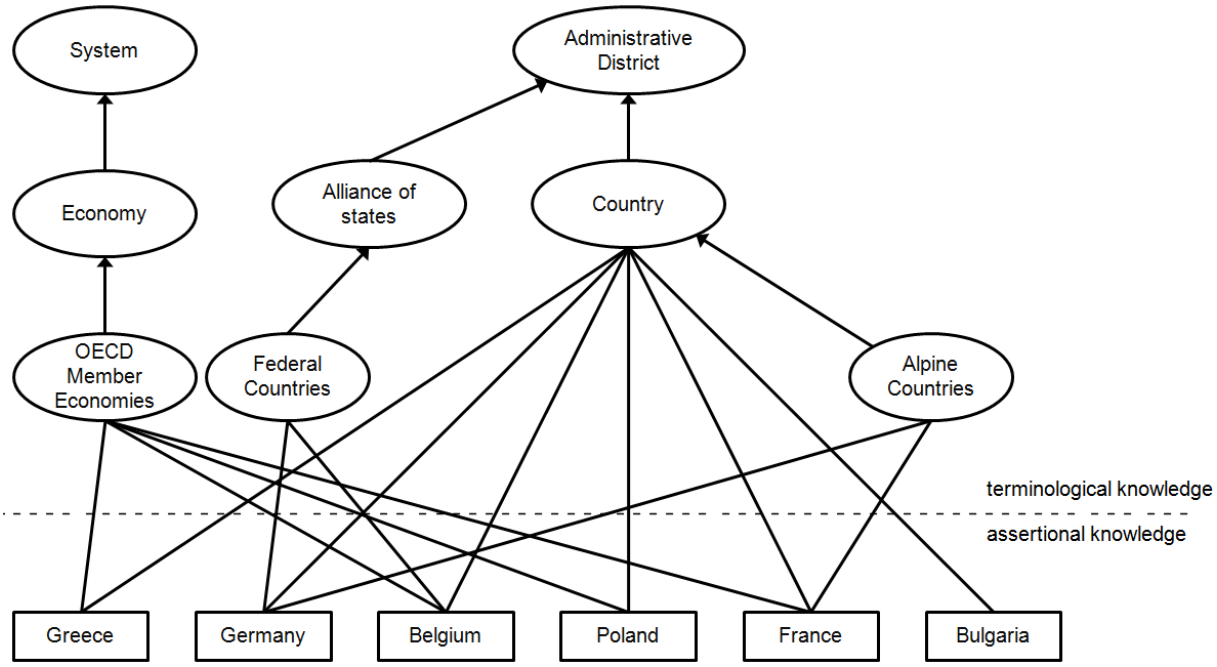

Figure 31: Simplified example for the approach 'rdfs:subClassOf': triples

At first it has to be noticed that in each case a parallel hierarchy is constructed, because there are two uppermost classes. The values of the parameters for this example are shown and the effect on the algorithm and the resulting hierarchy is

[45] http://dbpedia.org/About
[46] http://www.wikipedia.org/

described in the following. Because there is a basic distinction if the parameter for scheme is set or not, the hierarchy generation is explained for all two cases:

- Minimal number of stages: 2
  This means that the resulting hierarchy at least has two levels. This is here fulfilled, since three levels are derived.
- Maximal number of stages: 3
  This means that the resulting hierarchy at most has three levels. This is here fulfilled, since three levels are derived.
- Flag for the strictness: not set
  This means that a member can have several parent members. Here, for example Belgium has two parents.
- Flag for more members than parents: not set
  This means that it is allowed that a parent level contains more members than the child level. This is here not the case, since each time there are more members than parents.
- Flag for schema:
  - set:
    If the flag for schema is set, each class that is super class of another class results in an own level. Each sub class results as a member in the created level. For example the class ‘Alpine Countries’ results not in a level, because it has no sub class, but it results as a member in the class ‘Country’. The members ‘Germany’ and ‘France’ result as children of the member ‘Alpine Countries’. Because a member can only roll up to one level in a generalized hierarchy, one parent has to be chosen. This has for example to be done for the member ‘Germany’, where the parent ‘Federal Countries’ in the level ‘Alliance of states’ or the parent ‘Alpine Countries’ in the level ‘Country’ has to be chosen. The following hierarchy is then generated.

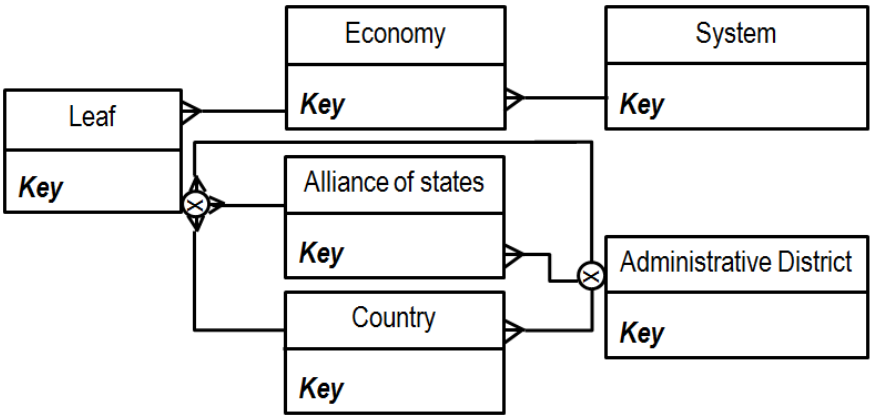

Figure 32: Resulting hierarchy of the example with setting the flag: schema

- not set:

  If the flag for schema is not set, each class results in a member and the sequence of classes determines the sequence of levels. The labels for the levels have to be determined generic, since no universal valid label can be found. For example the 'Alpine Countries' and 'Federal Countries' are members in 'Level 2 B', but no name could be found for this level. The following hierarchy is then generated.

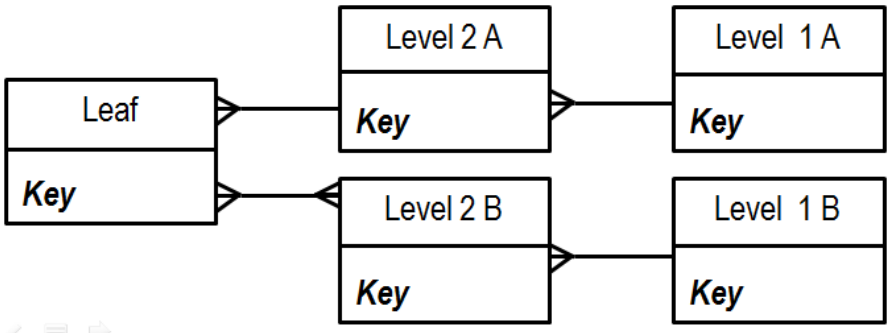

Figure 33: Resulting hierarchy of the example without setting the flag: schema

5.4.1.5 Resulting hierarchies

With this approach there may be generated almost all introduced OLAP hierarchies.

If there is one uppermost class, this results in a simple hierarchy. Otherwise, if there is more than one uppermost class, this results in a parallel hierarchy. The parallel hierarchy may be independent if no levels are shared or dependent, if levels are shared.

The included simple hierarchies are potential generalized, if the flag for schema is set, because generally speaking, each class results in an own level. If the flag for schema is not set, symmetric hierarchies are generated.

If the flag for strictness is set or there result only one parent per member, the hierarchy is strict. Otherwise the hierarchy is non-strict.

Asymmetric hierarchies cannot be found with this approach, since this approach is a bottom up approach. Also multiple alternative hierarchies cannot be found, because either there is generated a simple hierarchy if there is one uppermost level or a parallel hierarchy if there are several uppermost levels.

5.4.1.6 Criticism

This approach assigns all members of a given dimension of the data set(s) to one leaf level. This may not always correct. For this reason all members that should not be members of the leaf level have to be assigned to another level manually or in a further optimization step that is not part of this thesis.

Because a member can only roll up to one level in a generalized hierarchy, one parent has to be chosen if the flag for schema is set. This means when the algorithm is run several times with the same parameters, there may be constructed different hierarchies. As a consequence, the semantics is modified, because not all rdfs:subClassOf relationships are taken into account.

If the flag for schema is not set, the semantics is also modified when generating the hierarchy out of the ontology, because levels are constructed by the sequence of rdfs:subClassOf relationships between classes. This means that an instance of a class is treated the same way like a sub class. Labels of a level could not be determined using information out of the ontology, but have to be counted. This is why no universal valid label can be found for a certain level.

To ensure good results with this approach, the parameters could be set by default or some recommendations could be given. To give such a helpful recommendation, this approach could be applied on several data sets with several constellations of

parameters and the resulting hierarchies could be compared to each other, e.g. with help of the developed metrics.

5.4.2 **Approach 'properties'**

5.4.2.1 Overview

This approach, called 'properties', derives hierarchies from the used properties of the members. A property can be defined as a describing feature of a resource. The main idea is to use the three elements of a triple (subject, predicate and object) as elements of the hierarchy in a recursive way. The triples are regarding the members of a dimension which means that the member of a dimension is the subject of the existing triples.

As described in the RDF specification[47], the part that identifies the thing the statement is about is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies is called the predicate. The part that identifies the value of that property is called the object. The following table gives an overview of the mapping between existing element of a triple and resulting element of the hierarchy:

Triple	Hierarchy
Subject	Member of a level
Predicate/Property	Level above the subject level
Object	Parent member of the level above the subject level

Table 11: Mapping between triple and hierarchy of the approach 'properties'

5.4.2.2 Parameters

This approach is developed for at most general use. Theoretically, many possible hierarchies can result with this approach. The result set of possibly hierarchies of this approach can be influenced with the following parameters, which are in union termination criteria for the algorithm:

- Minimal number of stages:
  This parameter defines how many stages a resulting hierarchy has at least.

[47] http://www.w3.org/TR/rdf-primer/

- Maximal number of stages:
  This parameter defines how many stages a resulting hierarchy has at most.
- Flag for generalized or specialized properties:
  Since there is the possibility to define a specialization relationship between two properties via the property rdfs:subPropertyOf, only all uppermost (most generalized) properties are relevant, if this flag is set. As a consequence, the relevant properties are not sub property of any other property. If this flag is not set, only the bottommost (most specialized) properties are relevant. In this case the relevant properties do not have any sub property.
- Percentage of subjects that have a particular property:
  This parameter defines, how many percent of the subjects (members in a particular level) must have a particular property (parent level) so that this particular property results in a parent level. If the value of this parameter is 100%, only those properties are relevant which are properties of all members. In this case, a property which is not a property of all members is forming no level.
- Flag for the strictness:
  This parameter serves as decision criterion for the strictness of the resulting hierarchy. If this flag is set, the resulting hierarchy is strict. This may semantically be not correct, since there may be cases where a subject has several times the same property with different objects. This would result in several parent members. If this flag is set, one object is chosen to have only one parent member. If non-strictness is required, this flag has not to be set.
- Flag for more members than parents:
  If there would exist more parent members within one level than members, this would rather correspond to a drill down and not to a roll up. Because higher hierarchy levels should rather be reached by a roll up operation, more parents than members would not be reasonable in many cases. So if this flag is set, there have to exist more members than parent members within one level. If this is not the case, the actual property is forming no higher level.

5.4.2.3 Algorithm

5.4.2.3.1 Start

It is assumed that the members of a given dimension result in the lowest level of the hierarchy, so they are the subjects, from which this approach starts from. The following steps have then to be done:

5.4.2.3.2 Creating a concept scheme

To assign the created URIs to a concept scheme, for each of the above described simple hierarchies, a concept scheme has to be defined as follows. The given dimension of the given data structure definition(s) is then linked via qb:codeList to it:

```
hrccs:properties rdf:type skos:ConceptScheme.
```

As described above, the members of a dimension in the RDF Data Cube Vocabulary should be separated from the concepts of the classification hierarchy. Therefore, for each member of the given dimension in the given data sets, a new URI is created. The new URI can be orientated towards the URI of the member. For example for the URI <http://estatwrap.ontologycentral.com/dic/geo#DE> the new URI hrcco:estatwrap/ontologycentral/com/dic/geo/DE is created. Then, this is used as SKOS concept and assigned to the created concept scheme. The concept is then linked via rdfs:seeAlso to the original member:

```
hrcco:estatwrap/ontologycentral/com/dic/geo/DE rdf:type skos:Concept;
                skos:inScheme hrccs:Properties;
                rdfs:seeAlso <http://estatwrap.ontologycentral.com/dic/geo#DE>.
```

For constructing these triples, the following SPARQL construct query can be used. Hereby, the given data set(s) and dimension have to be filled in the filter statements and an URI for the resulting concept scheme has to be defined and filled in the bind statements (bold marked).

```
Construct{
#Creating a concept scheme
```

```
?concept rdf:type skos:Concept.
?concept rdfs:seeAlso ?member.
?concept skos:inScheme ?conceptScheme.
?conceptScheme rdf:type skos:ConceptScheme.
?dimension qb:codeList ?conceptScheme.
}
where{
?obs rdf:type qb:Observation.
?obs qb:dataSet ?dataset.
?dataset rdf:type qb:DataSet.
?obs ?dimension ?member.
?dataset qb:structure ?dsd.
BIND(URI(CONCAT("http://hierarchie.org/co/",
REPLACE(REPLACE(SUBSTR(STR(?member),8),"\\.","/"),"#","/"))) AS ?concept).

FILTER(
?dataset = <datasetURI1> ||
?dataset = <datasetURI2> ||
…
?dataset = <datasetURIn>
)
FILTER(
?dsd = <dsdURI1> ||
?dsd = <dsdURI2> ||
…
?dsd = <dsdURIn>
)
FILTER(?dimension = <dimensionURI>)
BIND(URI("conceptSchemeURI") AS ?conceptScheme).
}
```

5.4.2.3.3 Loading relevant triples

With the first run of this step, all triples are crawled of which the member and the resources that are linked to it with the owl:sameAs property are subject. This means triples are crawled from the URI of the member of the given dimension and to owl:sameAs resources of the member.

With each further iteration, additional concepts may have been assigned to the concept scheme in a higher level in the step before. The corresponding resources that are linked to the concepts via rdfs:seeAlso, are crawled, respecting owl:sameAs resources.

All these triples are added to the triple store.

5.4.2.3.4 Adding hierarchical information

Since this approach can have recursions, there has to be distinguished between the first and the other passes of this step. In each case, the depths of the levels are not set in this step, since there may be constructed more parents level with each further pass of this step. With each recursion, a new stage, meaning all levels with the same depth, is created.

At the first pass of this step, a leaf level is created. The concepts of the created concept scheme are assigned to this leaf level via skos:member. This level is assigned to the concept scheme via skos:inScheme:

```
hrccl:properties/leaf                          rdf:type          skosclass:ClassificationLevel;
                                               skos:inScheme     Properties.
hrcco:estatwrap/ontologycentral/com/dic/geo/DE
                                               skos:member       hrccl:properties/leaf.
```

The following description is valid for the first and all other passes of this step:

Each property can have several sub properties, which itself also can have sub properties. If the flag for generalized or specialized properties is set, only the uppermost (most generalized) properties are relevant. Consequently, a reasoning has to be done, whereby the rdfs:subPropertyOf relationship plays an important role. After applying the following rules, the members have all possibly properties, including the uppermost (most generalized properties):

```
[rdfs5:  (?x rdfs:subPropertyOf ?y), (?y rdfs:subPropertyOf ?z) -> (?x rdfs:subPropertyOf ?z)]
[rdfs6:  (?a ?p ?b), (?p rdfs:subPropertyOf ?q) -> (?a ?q ?b)]
```

These rules can be executed with the following SPARQL construct queries, whereby the construct query for the rule rdfs5 has to be executed several times until no additional triples are constructed:

```
Construct{
#rule rdfs5
?x rdfs:subPropertyOf ?z.
}
where{
```

```
?x rdfs:subPropertyOf ?y.
?y rdfs:subPropertyOf ?z.
}
```

```
Construct{
#rule rdfs6
?a ?q ?b.
}
where{
?a ?p ?b.
?p rdfs:subPropertyOf ?q.
}
```

This approach works stage by stage. With each passing of this step, an additional stage with several parent levels is created. Therefore, the parent levels, including parent members, of the levels of the actual stage have to be determined. Therefore, the following SPARQL Query can be run against the data, which has to be executed for each level of this stage separately. Hereby the URI of the actual level have to be filled in the brackets (bold marked). Some properties, e.g. owl:sameAs, rdfs:label, can be filtered out, because it makes no sense to use them as a parent level.

```
SELECT  distinct ?concept ?parentLevel ?parentConcept
#Determine candidates for the parent levels
WHERE {
?concept rdfs:seeAlso ?member.
?concept skosclass:member ?level.
?level rdf:type skosclass:ClassificationLevel.
OPTIONAL{?member owl:sameAs ?sameAsMember.}
{{?member ?property ?parent.} UNION {?sameAsMember ?property ?parent.}}
FILTER( ?property != owl:sameAs && ?property != rdfs:label )
FILTER( ?level = <levelURI> )
BIND(URI(CONCAT("http://hierarchie.org/co/",
REPLACE(REPLACE(SUBSTR(STR(?parent),8),"\\.","/"),"#","/"))) AS ?parentConcept).
BIND(URI(CONCAT("http://hierarchie.org/cl/",
REPLACE(REPLACE(SUBSTR(STR(?property),8),"\\.","/"),"#","/"))) AS ?parentLevel).
 [PROPERTY STATEMENT TO BE INSERTED HERE]
}
```

The following two property statements are possible:

- If the flag for generalized or specialized properties is set, the following statement has to be inserted, to retrieve only the uppermost properties:

```
FILTER NOT EXISTS {?parentLevel rdfs:subPropertyOf ?x}
```

- Otherwise, the following statement has to be inserted, to retrieve only the bottommost properties of a particular member:

```
FILTER NOT EXISTS {     ?x rdfs:subPropertyOf ?parentLevel.
                        ?member ?x ?parent.}
```

The query retrieves the following elements:

- Members of the actual level (?concept)
- Parent Levels (?parentLevel)
- Parents (?parentConcept)

These elements can be seen as candidates for elements of the hierarchy. It has to be proofed for each parent level, if the following conditions are fulfilled. If one of these conditions is not fulfilled, the particular property is forming no parent level:

- Percentage of subjects that have a particular property:
  It has to be proofed, if the percentage of subjects, which have a particular property that is forming a parent level, is greater or equal the value of the parameter. If not, the actual property is forming no parent level since there are too less subjects that have this particular property.
- No Loop:
  If the actual property is linking the subject back to a resource that is member of a lower level, the actual property is forming no parent level, since there may be no loops in an OLAP hierarchy.

As a consequence, each property that fulfills these conditions is forming a parent level. If the value of the parameter for the percentage of subjects that have a particular property is smaller than 100%, there may exist subjects that do not have this particular property, which is forming a parent level. Hence, they also have no object, which can be their parent in this parent level. For these reasons, all such

subjects are assigned to a dummy in the parent level, which serves as parent member in the parent level.

For each parent, concepts have to be created that are linked to the crawled resources via rdfs:seeAlso if the parent is a resource or rdfs:label if the parent is a label. This means that an URI is created for each different value of an occurring literal or resource, which is then defined as SKOS concept and assigned to the created concept scheme. If the flag for strictness is set, one parent is chosen randomly, if the property for the parent level is linked multiple times to the member. Also, for each property, levels have to be created, to which the created concepts are assigned to via skos:member. These levels are defined as skosclass:ClassificationLevel. Additionally, they have to be assigned to the scheme via the property skos:inScheme.

Furthermore, there have to be defined concept associations with skosclass:hasSourceConcept and skosclass:hasTargetConcept properties to child and parent of a relationship and an rdf:predicate link to skos:broader. This concept associations are also linked to the concept scheme via skos:inScheme.

For example, the query above retrieves the following triple, in which the property dbpedia:owl-language can be seen as a candidate for a parent level:

```
hrcco:estatwrap/ontologycentral/com/dic/geo/DE dbpedia-owl:language dbpedia:Germans.
```

Assuming that the two conditions are fulfilled, the following triples are created and added to the triple store:

```
hrcco:dbpedia/org/resource/Germans rdf:type                skos:Concept;
                                   skos:inScheme           hrccs:Properties;
                                   rdfs:seeAlso            dbpedia:Germans;
                                   skos:member             hrccl:dbpedia/org/ontology/language.
hrccl:dbpedia/org/ontology/language rdf:type               skosclass:ClassificationLevel;
                                   skos:inScheme           hrccs:Properties.
hrcca:dbpedia/org/resource/Germans_dbpedia/org/ontology/language
        rdf:type                                skosclass:ConceptAssociation;
        skosclass:hasSourceConcept              hrcco:dbpedia/org/resource/Germans;
```

rdf:predicate	skos:broader;
skosclass:hasTargetConcept	hrcco:dbpedia/org/resource/Germans;
skos:inScheme	hrccs:Properties.

5.4.2.3.5 End of the approach reached?

The end of the approach is reached, if one of the following termination criteria is fulfilled:

- Maximal number of stages exceeded:
  If the number of stages exceeds the maximal number of stages, no more parent level is created and the algorithm terminates.
- No additional levels added:
  If there were no additional levels added in the step before, the algorithm terminates.

Otherwise the algorithm continues with the second step, where triples to the members that have been assigned to the concept scheme beforehand, are crawled. The next iteration creates parent levels that are one stage above the stage that has been created beforehand. This means the algorithm works stage for stage. With each iteration, the next higher stage is created. In each stage there may be several parent levels.

5.4.2.3.6 End

At the end of the approach ‘properties’ the constructed concept scheme includes the hierarchical relationships which can then be used.

5.4.2.4 Example

There is a data set[48], which has the dimension eus:geo. The members of this dimension include owl:sameAs links to resources from dbpedia[49], a structured extract of wikipedia[50]. For illustration reasons, the following simplification reasons are made:

[48] http://estatwrap.ontologycentral.com/id/tsieb020#ds
[49] http://dbpedia.org/About

- Only six members are used
- The prefix dbpedia is excluded in the figure below
- Only the two following properties are used:
    - dbpedia-owl:language
    - dbpedia-owl:languageFamily

The following figure shows the resources and the two properties:

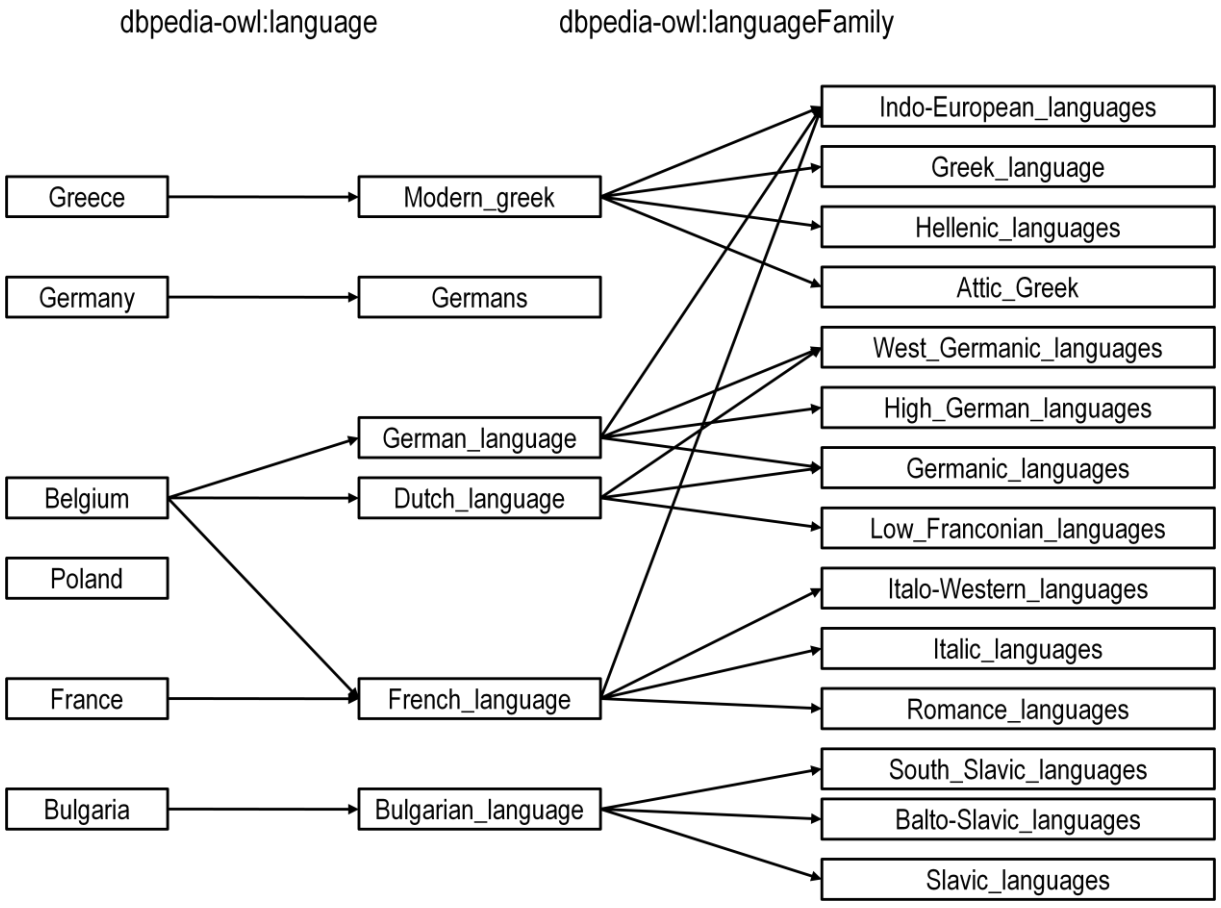

Figure 34: Simplified example for the approach 'properties': triples

At first, the algorithm independent of its parameters is explained: The first property dbpedia-owl:language results in the first level. Since Poland does not have this property, a dummy member is created in this level. This dummy is used as parent of all members that do not have the property dbpedia-owl:language. The second property dbpedia-owl:languageFamily results in the second level. Since Germans

[50] http://www.wikipedia.org/

does not have this property, also in this level a dummy member is created, again. This is a parent of all members that do not have the property dbpedia-owl:languageFamily.

There may be special cases, where a certain resource (e.g. Germanic_languages) is an object of the property owl:languageFamily and also an object of the property dbpedia-owl:language, which could be both properties of a member in the leaf level (e.g. Belgium). In this case, it has to be ensured that in each resulting simple hierarchy, this resource is assigned to only one level.

Now, the values of the parameters for this example are shown and the effect on the algorithm and the resulting hierarchy is described:

- Minimal number of stages: 2
  The resulting hierarchy at least has two levels. This is here fulfilled, since three levels are derived.
- Maximal number of stages: 3
  The resulting hierarchy at most has three levels. This is here fulfilled, since three levels are derived.
- Flag for generalized or specialized properties: not set
  This means that the rules for determining the uppermost properties are not applied. The already existing properties (dbpedia-owl:language and dbpedia-owl:languageFamily) instead of their possible super properties result in levels.
- Percentage of subjects that have a particular property: 0.6
  This means that 60% of the subjects must have a particular property so that this property results in a higher level. This is fulfilled, because 5 of 6 members have the property dbpedia-owl:language and 5 of 7 members have the property dbpedia-owl:languageFamily.
- Flag for the strictness: not set
  A member can have several parent members. Here, for example Belgium has three parents.
- Flag for more members than parents: not set
  This means that it is allowed that a parent level contains more members than the child level. This is here the case since the level owl:languageFamily has 14 members and the level dbpedia-owl:language has 7 members.

The resulting hierarchy in this example is a symmetric hierarchy, since all levels are populated with members. Furthermore, the resulting hierarchy is non-strict, since a member has several parents.

5.4.2.5 Resulting hierarchies

With this approach there may be generated almost all introduced OLAP hierarchies.

Each different uppermost level results in an individual hierarchy. If there is more than one uppermost level, this results in a parallel hierarchy. The parallel hierarchy may be dependent, if some levels are shared or independent otherwise. If there is only one uppermost level, the resulting hierarchy is individual.

If there are parallel levels on the same stage between the leaf and the uppermost level that are not part of a generalized hierarchy, this results in a multiple alternative hierarchy.

If the bottommost (most specialized) properties are used, there may result generalized and non-covering hierarchies. For each property that has several sub properties which are used for the members, an own generalized hierarchy is generated. Here, each property is forming one level at the same stage. If a member has a property and a sub property of this property, only the object of the bottommost property of this particular member results as a parent. If there are several sub properties of one particular property used for the members, only one of these sub properties results as parent level to guarantee exclusive paths in a generalized hierarchy. If there are several uppermost levels at the uppermost stage in a generalized hierarchy after executing the algorithm, all these uppermost levels are taken together in one common uppermost level to retrieve a proper generalized hierarchy.

Furthermore, the resulting simple hierarchies can potential be strict, if there is always used the same property once for a certain subject or if the parameter for strictness is set. Otherwise the hierarchy is non-strict. If the parameter for strictness is set, only one object is chosen as parent member, since a particular property can be used many times for the same subject.

Asymmetric hierarchies cannot be found with this approach, since this approach is a bottom up approach.

5.4.2.6 Criticism

With this approach, properties of a member result in parent levels of this member. However, properties could also be used to derive descriptive attributes of the members, where the object of a triple is the characteristic of this attribute. This means that if descriptive attributes are relevant for the resulting level, the properties could fulfill two functions, as level and descriptive attribute. If a property is used only once for a given member (which would result in a strict hierarchy) and the object of a triple fully functional depends on the subject of a triple, this is an indication of an attribute, since the value of this characteristic fully functional depends on the member. So this approach generates levels from properties instead of descriptive attributes, which should sometimes rather be modeled than levels.

Furthermore, this approach assigns all members of a given dimension of the data set(s) to one leaf level. This may not always correct. For this reason, all members that should not be members of the leaf level have to be assigned to another level manually or in a further optimization step that is not part of this thesis.

To ensure good results with this approach, the parameters could be set by default or some recommendations could be given. To give such a helpful recommendation, this approach could be applied on several data sets with several constellations of parameters and the resulting hierarchies could be compared to each other, e.g. with help of the developed metrics.

6 Evaluation

This section evaluates the developed approaches by testing them on real-world datasets from Eurostat[51]. Hereby the same datasets like in the examples of the particular approaches are used. The evaluation is based on the integration of two datasets. Therefore, two datasets are integrated for each approach.

6.1 Approach 'time'

To evaluate the approach 'time', the components of the following table are used:

Component	URI
data sets	http://estatwrap.ontologycentral.com/id/teilm020#ds
	http://estatwrap.ontologycentral.com/id/tsieb020#ds
data structure definitions	http://estatwrap.ontologycentral.com/dsd/teilm020#dsd
	http://estatwrap.ontologycentral.com/dsd/tsieb020#dsd
dimension	http://purl.org/dc/terms/date

Table 12: Components for the evaluation of the approach 'time'

After the approach 'time' is executed on these given components, a hierarchy with the following levels is constructed:

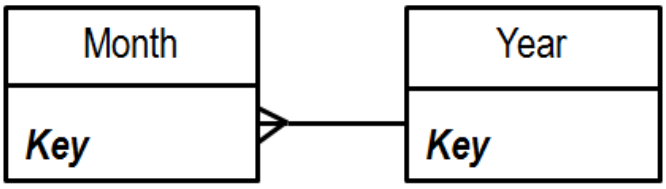

Figure 35: Resulting hierarchy of the approach 'time': schema

The integration of the two data sets is given via the member 2011, since the months 2011-04 till 2011-11, which occur in the first data set roll up to the year 2011, which is also included in the second data set. The resulting hierarchy is asymmetric, because not all members on in the level year have children in the level month and strict,

[51] http://estatwrap.ontologycentral.com/

because each member in the level months rolls up to only one parent member in the level year.

6.2 Approach ‘geo’

To evaluate the approach ‘geo’, the components of the following table are used:

Component	URI
data sets	http://estatwrap.ontologycentral.com/id/agr_r_landuse#ds
	http://estatwrap.ontologycentral.com/id/tsieb020#ds
data structure definitions	http://estatwrap.ontologycentral.com/dsd/agr_r_landuse#dsd
	http://estatwrap.ontologycentral.com/dsd/tsieb020#dsd
dimension	http://ontologycentral.com/2009/01/eurostat/ns#geo

Table 13: Components for the evaluation of the approach 'geo'

After the approach ‘geo’ is executed on these given components, a hierarchy with the following levels is constructed:

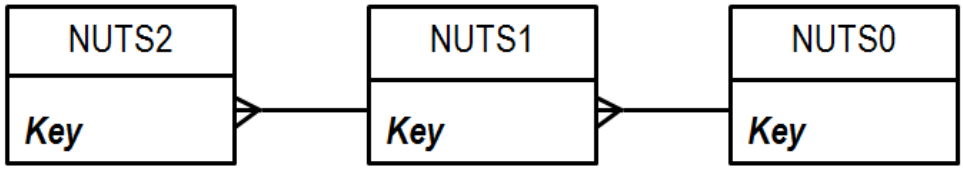

Figure 36: Resulting hierarchy of the approach 'geo': schema

The integration of the two data sets is given via the members of the level NUTS0, since some members of this level occur in both data sets and the members of the level NUTS1 roll up to some of these members of level NUTS0. The resulting hierarchy is asymmetric, because not all members in the level NUTS0 have children in the levels NUTS1 and NUTS2. Furthermore, the resulting hierarchy is strict, because each member in the level NUTS2 and NUTS1 rolls up to only one parent member.

6.3 Approach ‘rdfs:subClassOf’

To evaluate the approach ‘rdfs:subClassOf’, the components of the following table are used:

Component	URI
data sets	http://estatwrap.ontologycentral.com/id/cpc_siemp#ds
	http://estatwrap.ontologycentral.com/id/tsieb020#ds
data structure definitions	http://estatwrap.ontologycentral.com/dsd/cpc_siemp#dsd
	http://estatwrap.ontologycentral.com/dsd/tsieb020#dsd
dimension	http://ontologycentral.com/2009/01/eurostat/ns#geo

Table 14: Components for the evaluation of the approach 'rdfs:subClassOf'

There are 45 members in this given dimension, 29 members with owl:sameAs links and 16 members without owl:sameAs links to other resources. For the evaluation triples from dbpedia are used since there exist owl:sameAs links from members to resources from dbpedia. The parameters are set like in the example to the approach above and the parameter ‘Flag for schema’ is set. The following table gives an overview of the resulting levels on stage 2 which are above the leaf level and on stage 1 which are the uppermost levels.

Classes/Levels on stage 2	Property	Classes/Levels on stage 1
dbpedia-class: yago/Archipelago109203827	rdfs:sub ClassOf	dbpedia-class: yago/Land109334396
dbpedia-class: yago/AdministrativeDistrict108491826	rdfs:sub ClassOf	dbpedia-class: yago/District108552138
dbpedia-class: yago/System108435388	rdfs:sub ClassOf	dbpedia-class: yago/Group100031264
dbpedia-class: yago/Crisis113933560	rdfs:sub ClassOf	dbpedia-class: yago/SituAtion114411243
dbpedia-class: yago/Empire108557482	rdfs:sub ClassOf	dbpedia-class: yago/Domain108556491
dbpedia-class: yago/Capital108518505	rdfs:sub ClassOf	dbpedia-class: yago/Seat108647945
dbpedia-class: yago/Home108559508	rdfs:sub ClassOf	dbpedia-class: yago/Residence108558963
dbpedia-class: yago/Country108544813	rdfs:sub ClassOf	dbpedia-class: yago/AdministrativeDistrict108491826
dbpedia-class: yago/Location100027167	rdfs:sub ClassOf	dbpedia-class: yago/Object100002684
dbpedia-class: yago/Location100027167	rdfs:sub ClassOf	dbpedia-class: yago/YagoGeoEntity

dbpedia-class: yago/Location100027167	rdfs:sub ClassOf	dbpedia-class: yago/YagoLegalActorGeo
dbpedia-class: yago/Island109316454	rdfs:sub ClassOf	dbpedia-class: yago/Land109334396
dbpedia-class: yago/Region108630039	rdfs:sub ClassOf	dbpedia-class: yago/Location100027167
dbpedia-class: yago/Economy108366753	rdfs:sub ClassOf	dbpedia-class: yago/System108435388
dbpedia-class: yago/Clang107380144	rdfs:sub ClassOf	dbpedia-class: yago/Noise107387509
dbpedia-owl: PopulatedPlace	rdfs:sub ClassOf	dbpedia-owl: Place

Table 15: Levels of the approach 'rdfs:subClassOf'

After the approach ‘rdfs:subClassOf’ is executed on these given components, a hierarchy with the following levels is constructed. For illustration reasons, only the last seven levels on stage 1 of the above table are shown:

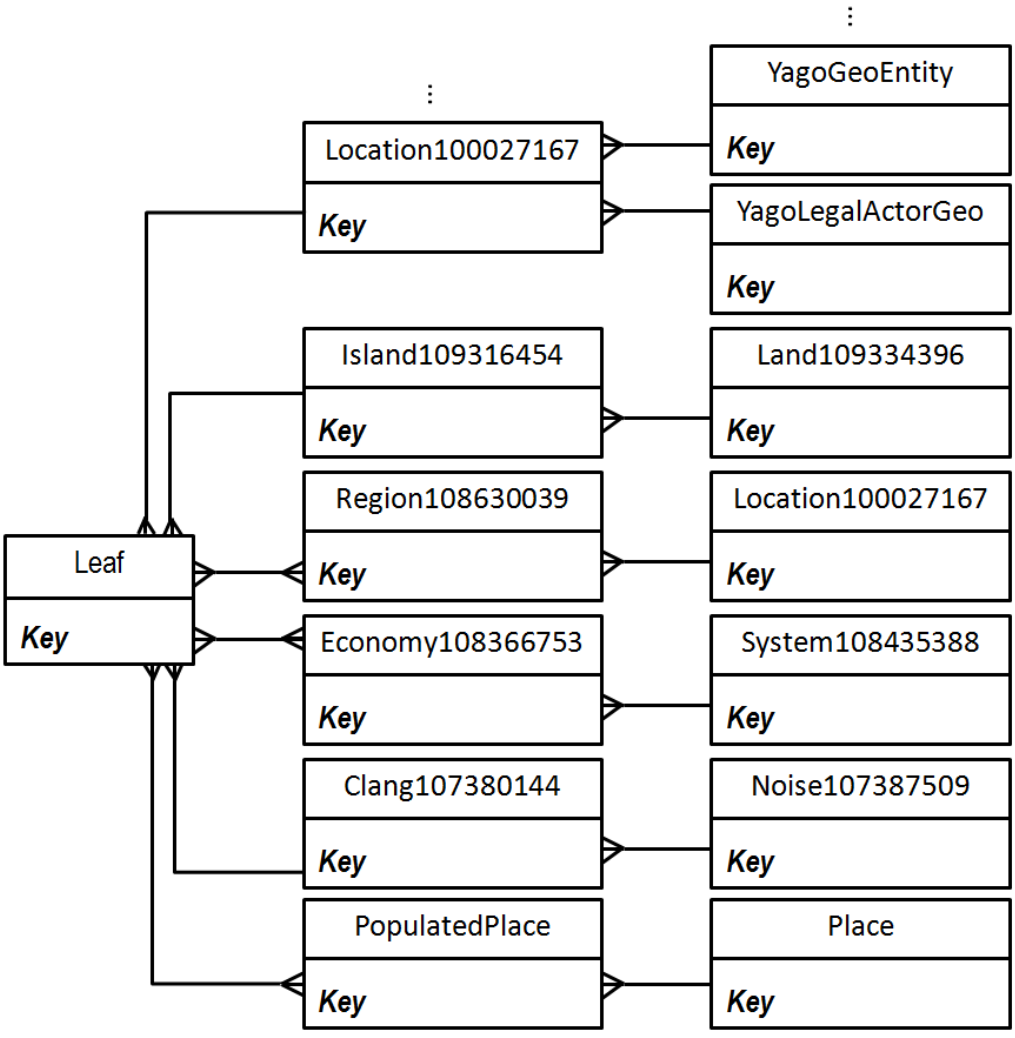

Figure 37: Resulting hierarchy of the approach 'subClassOf': schema

The integration of the two datasets is given via the members in the created levels on stage 2, since some members of this level are parents of members of both data sets. The resulting hierarchy is a parallel hierarchy, composed of several simple hierarchies. These simple hierarchies are symmetric, since all levels are populated with members. Depending on the number of rdf:type properties and rdfs:subClassOf relationships, the resulting simple hierarchies are strict or non-strict.

6.4 Approach ‘properties’

To evaluate the approach ‘properties’, the components of the following table are used:

Component	URI
data sets	http://estatwrap.ontologycentral.com/id/cpc_siemp#ds
	http://estatwrap.ontologycentral.com/id/tsieb020#ds
data structure definitions	http://estatwrap.ontologycentral.com/dsd/cpc_siemp#dsd
	http://estatwrap.ontologycentral.com/dsd/tsieb020#dsd
dimension	http://ontologycentral.com/2009/01/eurostat/ns#geo

Table 16: Components for the evaluation of the approach 'properties'

There are 45 members in this given dimension, 29 members with owl:sameAs links and 16 members without owl:sameAs links to other resources. For the evaluation triples from dbpedia are used since there exist owl:sameAs links from members to resources from dbpedia. The parameters are set like in the example to the approach above. For illustration reasons, only one stage above the leaf level is created. The following table gives an overview of the properties that are candidates for higher levels. Since the parameter ‘Percentage of subjects that have a particular property’ is set to 60%, only properties result in a parent level, where the distinct count of subjects is at least 27 members (0.6 * 45 members = 27 subjects).

Property	Distinct count of subjects having this property	Distinct count of objects
<http://dbpedia.org/property/leaderTitle>	28	74
<http://dbpedia.org/ontology/leaderName>	28	83
<http://dbpedia.org/ontology/capital>	28	28
<http://dbpedia.org/ontology/language>	27	70
<http://dbpedia.org/ontology/governmentType>	27	35
<http://dbpedia.org/ontology/currency>	27	29
<http://dbpedia.org/ontology/officialLanguage>	26	37
<http://dbpedia.org/property/timeZoneDst>	24	24
<http://dbpedia.org/property/timeZone>	24	24
<http://dbpedia.org/property/leaderName>	24	49
<http://dbpedia.org/property/capital>	24	24
<http://dbpedia.org/property/sovereigntyType>	22	22
<http://dbpedia.org/ontology/anthem>	22	25
<http://dbpedia.org/ontology/ethnicGroup>	21	67
<http://dbpedia.org/property/officialLanguages>	20	34
<http://dbpedia.org/property/governmentType>	19	26
<http://dbpedia.org/property/establishedEvent>	19	63
<http://dbpedia.org/property/demonym>	16	17
<http://dbpedia.org/property/legislature>	13	13
<http://dbpedia.org/property/currency>	11	12
<http://dbpedia.org/property/cctld>	11	12
<http://dbpedia.org/ontology/largestCity>	9	9
<http://dbpedia.org/property/upperHouse>	8	8
<http://dbpedia.org/property/lowerHouse>	8	8
<http://dbpedia.org/ontology/regionalLanguage>	7	20
<http://dbpedia.org/property/nationalAnthem>	4	4
<http://dbpedia.org/property/religion>	2	2
<http://dbpedia.org/property/regionalLanguages>	2	3
<http://dbpedia.org/property/callingCode>	2	2
<http://dbpedia.org/property/stateLanguage>	1	1
<http://dbpedia.org/property/sovereigntyNote>	1	1
<http://dbpedia.org/property/otherSymbolType>	1	1
<http://dbpedia.org/property/nonOfficialLanguages>	1	1
<http://dbpedia.org/property/nationalTree>	1	1
<http://dbpedia.org/property/nationalPoet>	1	1
<http://dbpedia.org/property/nationalPlant>	1	1
<http://dbpedia.org/property/nationalMotto>	1	1
<http://dbpedia.org/property/nationalLanguage>	1	1
<http://dbpedia.org/property/nationalBird>	1	1
<http://dbpedia.org/property/largestSettlement>	1	1
<http://dbpedia.org/property/largestCity>	1	1
<http://dbpedia.org/property/languagesType>	1	1
<http://dbpedia.org/property/languages>	1	1
<http://dbpedia.org/property/frMetropole>	1	1
<http://dbpedia.org/property/establishedDate>	1	1
<http://dbpedia.org/property/caption>	1	1
<http://dbpedia.org/ontology/wikiPageRedirects>	1	1
<http://dbpedia.org/ontology/largestSettlement>	1	1

Table 17: Candidates for levels of the approach 'properties'

When crawling these triples, the following properties have not been considered, since they are not very useful to generate hierarchies or they are used in another approach. For example rdf:type is used in the approach ‘rdfs:subClassOf’ and <http://xmlns.com/foaf/0.1/page> links to a homepage:

- <http://www.w3.org/2002/07/owl#sameAs>
- <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
- <http://www.w3.org/2000/01/rdf-schema#comment>
- <http://www.w3.org/2000/01/rdf-schema#label>
- <http://dbpedia.org/ontology/abstract>
- <http://dbpedia.org/ontology/wikiPageExternalLink>
- <http://xmlns.com/foaf/0.1/homepage>
- <http://xmlns.com/foaf/0.1/page>
- <http://xmlns.com/foaf/0.1/depiction>
- <http://dbpedia.org/property/wikiPageUsesTemplate>
- <http://dbpedia.org/ontology/thumbnail>
- <http://purl.org/dc/terms/subject>

After the approach ‘properties’ is executed on these given components, a hierarchy with the following levels on the first stage is constructed:

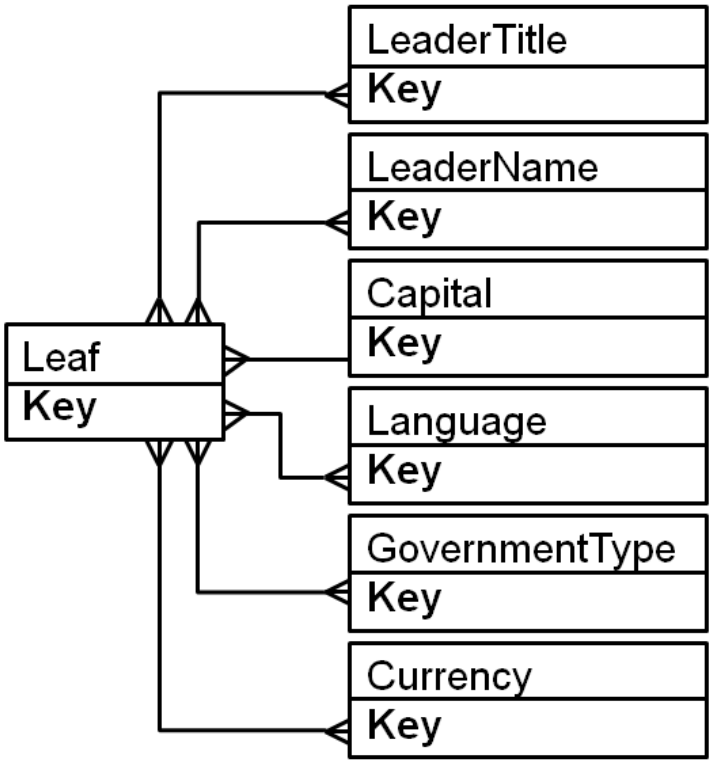

Figure 38: Resulting hierarchy of the approach 'properties': schema

The integration of the two datasets is given via the members in the created levels on stage 1, since some members of this level are parents of members of both data sets. The resulting hierarchy is a parallel hierarchy, composed of several simple hierarchies. These simple hierarchies are symmetric, since all levels are populated with members and there are no rdfs:subPropertyOf relationships between the properties. Furthermore, the resulting simple hierarchies that have more parents than members are non-strict and the simple hierarchy leaf→capital is strict, because this property is used only once for a subject.

7 Conclusions

Upon finishing the main parts of this thesis, this last section summarizes them, describes the resulting lessons learned and looks ahead for possible future research topics.

7.1 Summary

This master thesis showed how hierarchies can be constructed from Statistical Linked Data and be used in OLAP systems. After a general introduction and explanation of the used technologies, it answered the following main questions:

- Which are useful Statistical Linked Data hierarchies?
  This question was answered in section 2 ‘Theoretical background’, where the relevant concepts, technologies and standards, especially the conceptual model of OLAP hierarchies, have been introduced. Furthermore, the developed metrics for measuring the usefulness of a hierarchy was described.
- How can Statistical Linked Data hierarchies be expressed?
  A solution on this question was given in section 3 ‘Expressing OLAP hierarchies in RDF’, where SKOS and the proposed extension have been explained so that all hierarchies of the conceptual model can be expressed with RDF.
- How can Statistical Linked Data hierarchies be used?
  The answer on this question was explained in section 4 ‘Transforming Linked Data into OLAP hierarchies’, where OLAP4J for the use in OLAP clients have been introduced. There have been developed SPARQL queries, which provide the content for the variables of the hierarchy relevant OLAP4J methods.
- How can Statistical Linked Data hierarchies be constructed?
  This question was answered in section 5 ‘Approaches for learning OLAP hierarchies from RDF’, where the different approaches to generate useful hierarchies have been explained. There have been developed specific approaches for temporal and geographical dimensions and generic approaches for potential all dimensions. The following table shows, which hierarchy can potentially be constructed with each approach.

Hierarchy			Approach			
			time	geo	rdfs: subClassOf	properties
parallel independent			-	-	X	X
parallel dependent			-	-	X	X
multiple alternative			X	-	-	X
simple	strict	symmetric	X	X	X	X
		asymmetric	X	X	-	-
		generalized (except non-covering)	-	-	X	X
		non-covering	-	-	X	X
	non-strict	symmetric	X	-	X	X
		asymmetric	X	-	-	-
		generalized (except non-covering)	-	-	X	X
		non-covering	-	-	X	X

Table 18: Resulting hierarchies of each approach

7.2 Resulting lessons learned

While working on this thesis, the following experiences have been made for constructing OLAP hierarchies from Statistical Linked Data. They can be seen as lessons learned for future research:

- Proper vocabulary required:

  There has been much effort to develop and propose the extensions for SKOS to have the possibility to express all types of hierarchies and distinguish different hierarchies from each other. As a consequence, the time for developing the approaches to generate hierarchies was reduced, which was the most important part of this thesis. This means that a proper vocabulary is an essential requirement, before the approaches to generate hierarchies could be developed. If such a proper vocabulary had already been existed, more effort would have been left over for developing, evaluating and optimizing the approaches.

- Different aggregation level:

  Data publisher often publish values on different aggregation level altogether in one data set, but do not express the included hierarchies. For example the

dimension eus:sex of a dataset[52] includes members for females[53], males[54] and total[55]. Since no hierarchical relationships between females/males and total is given, a data consumer or service provider has to handle this problematic, e.g. by filtering on female and male. This master thesis recommends to publish only observations on the most granular level and to define hierarchies. With the help of aggregation functions, which are not part of this thesis, and hierarchies, data consumer or service provider are able to compute the aggregated value theirselves. Hence, replicability is ensured and misinterpretations are avoided. Possible other methods to handle this problematic are the following:

- Introducing a parameter:
  There could be defined a flag that indicates the aggregation level of the members in the given data sets. If such a flag is set, all given members would result in the same level. If such a flag is not set, members of a given dimension of a dataset should be assigned to different levels.
- Specifying the most granular values:
  The members on the most granular level could be specified for example by a human in a separate preprocessing step before the approaches could be executed. All specified members would result in the leaf level.
- Extending the RDF Data Cube Vocabulary:
  All members on the most granular level in a certain data set could be marked with an extension of the RDF Data Cube Vocabulary. All marked members would result in the leaf level.

- Data quality:
  Data quality is an important success factor for finding useful hierarchies. If there are incorrect triples, which should be used to find hierarchies, also the resulting hierarchy can be wrong. Incorrect triples means that wrong information is included in a triple on the semantic level. For example, the following triple is wrong, saying that a language in Germany is the German people:

[52] estatwrap.ontologycentral.com/id/tsiem020#ds
[53] http://estatwrap.ontologycentral.com/dic/sex#F
[54] http://estatwrap.ontologycentral.com/dic/sex#M
[55] http://estatwrap.ontologycentral.com/dic/sex#T

dbpedia:Germany	dbpedia-owl:language	dbpedia:Germans.

The right triple would be the following, saying that the German language is a language in Germany:

dbpedia:Germany	dbpedia-owl:language	dbpedia:German_language.

This wrong triple would for example lead to a wrong parent member in the approach 'properties' in a hierarchy country→language.

- Technical problems:
  When developing the approaches, appearing technical problems have to be handled. For example when queries should be executed or triples should be added to the triple store the following problem appeared: The RDF/XML information resource[56] of a certain URI[57] was not XML well-formed, which means that this is not an RDF document and the triples could not be used.
- Hard- and software:
  Hard- and software is needed for developing the approaches. Mainly a proper triple store is required, where the approaches could be tested. Since not all triple stores provide SPARQL1.1[58] functionalities, which were needed for the developed SPARQL queries, this master thesis had to use a triple store, which fulfills this requirement. This was done on an Open RDF Sesame Server[59] in version 2.6.2.
- OLAP driver:
  To use the constructed hierarchies in OLAP systems, an OLAP driver is required, which transforms the resulting triples for the use in OLAP applications. This master thesis makes use of OLAP4 as OLAP application. Because of expressing hierarchies in Linked Data, also other OLAP applications could potential be supported. In the case of XMLA, the supported hierarchies are the same as in OLAP4J. This means that using Linked Data for expressing statistical data provides the possibility to support a broad range of applications and technologies.

[56] http://dbpedia.org/data/German_language
[57] http://dbpedia.org/resource/German_language
[58] http://www.w3.org/TR/sparql11-query/
[59] www.openrdf.org/

7.3 Future research topics

Hierarchies in the context of OLAP and Statistical Linked Data have been the focus of this master thesis. For doing further research in this area, the following topics are possibly the most relevant ones:

- Aggregation functions:
  As described in section 2.1.1.2 ‘Aggregation functions and summarizability’, this master thesis does not make a statement of aggregating the measures in a roll-up operation along a constructed hierarchy. Since the main focus of OLAP systems is to derive knowledge out of data, aggregated data is potential more interesting than the single granular values. For this reason, besides the need of hierarchies, research in aggregating values with use of these hierarchies can surely be seen as the next major work.
- Performance:
  If useful ways are found to aggregate the data with help of the generated hierarchies, performance will surely play in important role, when end users analyze Statistical Linked Data. To reduce the response times, research for improving the performance will positively influence the relevance of Statistical Linked Data. Therefore roll-up operations could be executed every time the data changes and the aggregated values could be saved. This concept is called small materialized aggregates (SMAs) [Moer98]. Possibly extending the RDF Data Cube Vocabulary and SKOS, it would be interesting to transfer this concept to Statistical Linked Data.
- Storage technologies:
  As described in section 2.1.3.2 ‘Star and snowflake schema’, there is a difference between the semantic, conceptual level of modeling multidimensional data and the logical, intern level of the database. Since Statistical Linked Data is in format of RDF, which is constructed out of triples, research in storage technologies for especially RDF triples would possibly be interesting to influence the performance. Therefore the existing technologies relational OLAP (ROLAP), multidimensional OLAP (MOLAP) and hybrid OLAP (HOLAP) could influence the storage of RDF triples. Also buffering triples in the main memory could be of research interest to enhance the response time.

- Approaches:

  Besides the other main parts of this master thesis, how hierarchies can be expressed and used, generating hierarchies is also a main part of this thesis. Further research in generating hierarchies could consist of improving and extending the developed approaches, especially the generic ones. Furthermore, there could be developed further approaches to generate hierarchies. The following ideas show, how possibly further approaches could be look like to find useful OLAP hierarchies:

  - Machine learning approaches:
    - Clustering:

      With the help of existing triples, in which the members of a dimension are part in the subject or object part, the members could be clustered.
    - Classification:

      With the help of triples, the members could be classified. For example if the range of a dimension is a literal with a numeric data type, e.g. xsd:integer, there could be defined thresholds statically or dynamically and the members could be classified in different classes in the sense of classification, e.g. small, medium, large. The resulting hierarchy would consist of two levels, one for the members one for the classes, in which the members were classified.
  - Human-centered approaches:
    - MDX-Query:

      If a user queries data, MDX queries are created. Hierarchies could possibly be found with the help of them, by using the required levels that are part of the MDX query.
    - Gaming:

      With the help of mini online games, human users could possibly optimize or construct hierarchical structures, which could then be used for OLAP.
  - Semantic approaches:
    - Special properties:

There could be identified special properties, which already represent hierarchical structures, for example the properties dbprop:fam and dbprop:child. They could be used to derive hierarchies.

- rdfs:subPropertyOf:
  Since there can be defined hierarchical structures between properties by using rdfs:subPropertyOf, this relationships can potentially be used, if there exist triples to a member, where these properties are included.

- Transforming approaches:
  - Already existing SKOS concept schemes:
    There may potentially exist SKOS concept schemes, where relationships are defined by skos:broader and skos:narrower, but the proposed extensions, e.g. skosclass:ClassificationLevel are not used. These concept schemes can be extended, so that they can be used by the developed SPARQL queries for the OLAP4J methods.
  - Other vocabularies:
    Besides SKOS, there may exist or be developed other vocabularies for expressing thesauri or other hierarchical structures, which could possibly be transformed in the extended SKOS vocabulary.

- Change of hierarchies:
  Although the approaches are parameterized equally in two runs of an approach, there may be generated different hierarchies, because additional triples could be found on the web, which could be used for adding hierarchical information to a certain concept scheme or the triples in the given data sets and data structure definitions may change. For this reason, a comprehensive change management of hierarchies is required.
- Usage:
  Usage means both, on the one side usage of the approaches to generate hierarchies and on the other side usage of the hierarchies to derive new knowledge.

- Usage of the approaches:

  To benefit from the developed approaches, they have to be provided for other users and applications in a way that hierarchies could be constructed. With use of the introduced possibility to provide a web service, the resulting triples could be added to a triple store, where OLAP applications can use them.

- Usage of the hierarchies:

  Since hierarchies can be used to integrate and analyze data on different aggregation level, intelligent strategies and concepts could be developed to finally use hierarchies for data on different aggregation levels in innovative applications, e.g. agents for decision support or recommender systems.

References

[AnAS04]
Anandarajan, M.; Anandarajan, A.; Srinivasan, C.: Business Intelligence Techniques: A Perspective from Accounting and Finance, 1. Auflage, Springer, Berlin, 2004.

[Bern06]
Berners-Lee, T. Linked Data - Design Issues, 2006, Retrieved 2011-06-02 from http://www.w3.org/DesignIssues/LinkedData.html

[Brat07]
Bratt, S.: Semantic Web, and Other Technologies to Watch, 2007, Retrieved 2011-06-04 from http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)

[BaGü04]
Bauer, A.; Günzel, H.: Data Warehouse Systeme – Architektur, Entwicklung, Anwendung, 2. edition, dpunkt, Heidelberg, 2004.

[BiHB09]
Bizer, C.; Heath, T.; Berners-Lee, T.: Linked Data – The story so far. International Journal on Semantic Web and Information Systems (IJSWIS), 2009, Volume 5(3), pp. 1-22.

[BeHL01]
Berners-Lee, T.; Hendler, J.; Lassila, O.: The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, Scientific American, 2001, Volume 284 (5), pp. 34–43.

[CoCS93]
Codd, E.; Codd, S.; Salley, C.: Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate. White Paper, 1993.

[ChDa97]
Chaudhuri, S.; Dayal, U.: An overview of data warehousing and OLAP technology. In: ACM SIGMOD Record, 1997, Volume 26 Issue 1, pp. 65 – 74.

[ChGl06]
Chamoni, P.; Gluchowski, P.: Entwicklungslinien und Architekturkonzepte des On-Line Analytical Processing. In: Chamoni, P.; Gluchowski, P. (Hrsg.): Analytische

Informationssysteme - Business Intelligence-Technologien und –Anwendungen, 3. Auflage, Springer, Berlin, 2006, pp. 143 – 176.

[CGSH10]
Correndo, G.; Granzotto, A.; Salvadores, M.; Hall, w.; Shadbolt, N.: A Linked Data representation of the Nomenclature of Territorial Units for Statistics, 2010, Retrieved 2011-10-21 from http://linkeddata.future-internet.eu/images/f/f0/FIA2010_A_Linked_Data_representation_of_the_Nomenclature_of_Territorial_Units_for_Statistics.pdf

[CyJe11]
Cyganiak, R.; Jentzsch, A.: The Linking Open Data cloud diagram, 2011, Retrieved 2011-10-26 from http://lod-cloud.net/

[CyRT10]
Cyganiak, R.; Reynolds, D.; Tennison, J.: The RDF Data Cube vocabulary, 2010, Retrieved 2011-09-23 from http://publishing-statistical-data.googlecode.com/svn/trunk/specs/src/main/html/cube.html

[FBSV00]
Franconi, E.; Baader, F.; Sattler, U.; Vassiliasis, P.: Multidimensional Data Models and Aggregation. In: Jarke, M.; Lenzerini, M; Vassiliou, Y.; Vassiliadis, P. (Hrsg.): Fundamentals of Data Warehouses, Springer, Berlin, 2000, pp. 87-105.

[Gart10]
Gartner, Inc.: User Survey Analysis: Key Trends Shaping the Future of Data Center Infrastructure Through 2011, 2010, Retrieved 2011-10-25 from http://www.gartner.com/resId=1456135

[GlGD08]
Gluchowski, P.; Gabriel, R.; Dittmar, C.: Management Support Systeme und Business Intelligence – Computergestützte Informationssysteme für Fach- und Führungskräfte. 2. Auflage, Springer, Berlin, 2008.

[GaGP09]
Gabriel, R.; Gluchowski, P.; Pastwa, A.: Data Warehouse & Data Mining, W3L, Witten-Herdecke, 2009.

[Humm08]
Hummeltenberg, W.: Business Intelligence, Retrieved 2011-06-05 from http://www.oldenbourg.de:8080/wi-enzyklopaedie/lexikon/daten-wissen/Business-Intelligence

[HiKR09]

Hitzler, P.; Krötzsch, M.; Rudolph, S.: Foundations of Semantic Web Technologies, Chapman & Hall/CRC, Boca Raton, 2009.

[Inmo02]
Inmon, W.: Building the Data Warehouse, 3. edition, John Wiley & Sons, New York, 2002.

[KaBM08]
Kashyap, V.; Bussler, C.; Moran, M.: The Semantic Web – Semantics for Data and Services on the web, Springer, Berlin, 2008.

[KäHa11]
Kämpgen, B.; Harth, A.: Transforming Statistical Linked Data for Use in OLAP Systems, 2011, Retrieved 2011-10-26 from http://www.aifb.kit.edu/images/2/28/Kaempgen_harth_isem11_olap.pdf

[KeMU06]
Kemper, H.; Mehanna, W.; Unger, C.: Business Intelligence: Grundlagen und praktische Anwendung: Eine Einführung in die IT-basierte Managementunterstützung, 2. Auflage, Friedr. Vieweg & Sohn, Wiesbaden, 2006.

[LeSh97]
Lenz, H. J.; Shoshani, A.: Summarizability in OLAP and statistical data bases. In: Proceedings Ninth International Conference on Scientific and Statistical Database Management Cat No97TB100150, IEEE Computer Society, 1997, pp. 132-143.

[LeTh09]
Lenz, H. J.; Thalheim, B.: A Formal Framework of Aggregation for the OLAP-OLTP Model. In: Journal of Universal Computer Science, Volume 15 Issue 1, Verlag der Technischen Universität Graz, Graz, 2009, pp. 273-303.

[Moer98]
Moerkotte, G.: Small materialized aggregates: A light weight index structure for data warehousing, 1998, Retrieved 2012-02-10 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.7211&rep=rep1&type=pdf

[MaZi04]
Malinowski, E.; Zimányi, E: OLAP Hierarchies: A Conceptual Perspective. In: Lecture Notes in Computer Science, Volume 3084/2004, Springer, Berlin, 2004, pp.19-35.

[MaZi05]

Malinowski, E.; Zimányi, E.: Hierarchies in a multidimensional model: From conceptual modeling to logical representation. In: Data & Knowledge Engineering 59, Frankfurt, 2006, pp. 348-377.

[Nort05]
North, K: Wissensorientierte Unternehmensführung: Wertschöpfung durch Wissen, 4. Auflage, Gabler, Wiesbaden, 2005.

[NiNi10]
Niemi, T.; Niinimäki, M.: Ontologies and summarizability in OLAP. In: Proceedings of the 2010 ACM Symposium on Applied Computing SAC 10, ACM Press, New York, 2010, p.1349.

[NiNT01]
Niemi, T.; Nummenmaa, J.; Thanisch, P.: Constructing OLAP cubes based on queries. In: Proceedings of the 4th ACM international workshop on Data warehousing and OLAP DOLAP 01, ACM Press, New York, 2001, pp. 9-15.

[OrDD06]
Oren,E.; Delbru, R.; Decker, S.: Extending Faceted Navigation for RDF Data. In: Lecture Notes in Computer Science, Volume 4273/2006, Springer, Berlin, 2006, pp. 559-572.

[PBAP08]
Perez Marti, J.; Berlanga, R.; Aramburu, M.; Pedersen, T.: Integrating Data Warehouses with Web Data: A Survey. IEEE Transactions on Knowledge and Data Engineering, 2008, pp. 940-955.

[PaMa11]
Pardillo, J.; Mazón, J.-N.: Using Ontologies for the Design of Data Warehouses, 2011, Retrieved 2011-10-15 from http://arxiv.org/ftp/arxiv/papers/1106/1106.0304.pdf

[PoRa99]
Pourabbas, E.; Rafanelli, M.: Characterization of hierarchies and some operators in OLAP environment. In: DOLAP99 Proceedings of the 2nd ACM International Workshop on Data Warehousing and OLAP (ACM), 1999, pp. 54-59.

[Sack10]
Sack, H.: Semantische Suche – Theorie und Praxis am Beispiel der Videosuchmaschine yovisto.com. In: Hengartner, U.; Meier, A. (Hrsg.): Web 3.0 & Semantic Web, dpunkt.verlag, Heidelberg, 2010, pp. 13-25.

[Vass98]

Vassiliadis, P: Modeling Multidimensional Databases, Cubes and Cube Operations. In: Proceedings of the 10th SSDBM Conference, IEEE Computer Society, Washington, 1998, pp. 53-62.

[Wagn10]
Wagner, G.: Zensus 2010/11 – eine längst überfällige Erhebung. In: Wochenbericht des DIW Berlin, 2010, Nr. 4/2010, pp. 11-14.

[ZaHM11]
Zapilko, B.; Harth, A.; Mathiak, B.: Enriching and Analysing Statistics with Linked Open Data. In: Eurostat (Hrsg.): NTTS - Conference on New Techniques and Technologies for Statistics, Brüssel, 2011.

Appendix: Namespaces

Prefix	Namespace
dbpedia	http://dbpedia.org/resource/
dbpedia-class	http://dbpedia.org/class/
dbpedia-owl	http://dbpedia.org/ontology/
dbprop	http://dbpedia.org/property/
dcterms	http://purl.org/dc/terms/
eus	http://ontologycentral.com/2009/01/eurostat/ns#
ex	http://example.org/
hrc	http://hierarchie.org/
hrcca	http://hierarchie.org/ca/
hrccl	http://hierarchie.org/cl/
hrcco	http://hierarchie.org/co/
hrccs	http://hierarchie.org/cs/
interval	http://reference.data.gov.uk/def/intervals/
nuts	http://nuts.psi.enakting.org/id/
nutsdef	http://nuts.psi.enakting.org/def/
owl	http://www.w3.org/2002/07/owl#
qb	http://purl.org/linked-data/cube#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
refgovukday	http://reference.data.gov.uk/id/gregorian-day/
refgovukmonth	http://reference.data.gov.uk/id/gregorian-month/
refgovukweek	http://reference.data.gov.uk/id/gregorian-week/
refgovukyear	http://reference.data.gov.uk/id/gregorian-year/
skos	http://www.w3.org/2004/02/skos/core#
skosclass	http://ddialliance.org/ontologies/skosclass#
spatial	http://data.ordnancesurvey.co.uk/ontology/spatialrelations/
time	http://www.w3.org/2006/time#
xsd	http://www.w3.org/2001/XMLSchema#